UNIVERSITY OF BURGOS

Area of Electronic Technology

Parallel & Hybrid Programming.

José María Cámara Nebreda, César Represa Pérez, Pedro Luis Sánchez Ortega
Parallel & Hybrid Programming. 2015
Area of Electronic Technology
Electromechanical Engineering Department
University of Burgos

Introduction

Our interest will be focused on parallel programing for multicomputer MIMD machines. Our application programs will split into several processes and each one will have the potential capability to be executed on a different node of our cluster.

The processes created by the user will cooperate to achieve a common computational objective. The collaboration will be possible due to communication and synchronization tools provided by the programming environment. Communication is implemented in the form of message exchanging.

Most of the scenarios proposed admit a number of different parallel solutions. We should try to come up with the most advantageous in terms of system performance. To do so we must take into account:

- We will try to increase performance (execution time). To do so, we will try to squish the application's potential locality, that is, its capability to work with local data avoiding the need for much information exchange between processes.
- Another important point is "scalability". In a hardware environment, where the amount of available resources is unknown at programming time, the application must scale to make the most of the available resources at any time.

Parallel programing is not an easy job. The theory around the development of concurrent and parallel software is beyond the scope of this course but, we will provide some hints. Parallel programming, as well as sequential programming is a creative task; what is about to be exposed is nothing more than a series of steps we recommend to follow when facing a parallelization. Let's split up the process in 4 steps:

- **Fragmentation**: this initial step is meant to find potential parallel structures within the problem to be solved. As a first approach, we may try to decompose the job in as many small parallel tasks as possible. Two criteria can be followed to carry out this decomposition:
  - The functional way: seeks for possible divisions in the job to be carried out by paying attention to its nature.
  - The data way: pays attention to the nature of the data to be processed trying to decompose them into the smallest chunks.
- **Communication**: once identified potential parallel tasks, communication needs between them must be analyzed.
- **Binding**: given that the cost of communications is high in terms of global execution time, the formerly identified tasks have to merge partially in order to balance computation and communication.
- **Mapping**: once the program's structure is settled, the recently generated processes have to be spread across the computers available. The strategy to be adopted differs according to

the fragmentation way. As a rule of thumb, there should be at least as many processes as computers are available in order to prevent anyone being unused. If all computers are equal, it would be recommendable to make as create as many processes as computers. If not, the most powerful computers can host a higher number of processes. It is also possible to assign processes to nodes on the go, thus balancing processors' load dynamically.

Depending on several aspects, being the type of computer one of the most relevant, parallel programming admits different approaches:

- Message passing: especially indicated for distributed memory computers, can be used on any hardware platform.
- Shared memory: suitable only for shared memory environments.
- Hybrid programming: a combination of the two previous. It is meant to optimize performance when both shared and distributed memory schemes are present. This scenario is very common in recent days. Modern clusters and MPPs are integrated by multicore memory sharing nodes.

In this course we will assume that the student is familiar enough with message passing programming. More precisely, the concepts given in the Bachelor Degree on Computer Science about MPI programming are considered as known. Otherwise it is highly recommended for the student to go through the MPI Programming Fundamentals course. At least from activity 0 to activity 5.

Activity 1: MPI Matrix Multiplication

OBJETIVES

- Apply previously acquired knowledge to develop a bit more complex program intended to be used as a benchmark to measure system performance.

THEORETICAL CONCEPTS

No new concepts will be introduced in this chapter since it is meant to exploit those already learned. As obvious, not all aspects of MPI development environment have been exposed and nor our application program is expected to find the most optimal solution but quite a good job is possible though.

However, it may be helpful to introduce some additional information about the functions we already know. Function **`MPI_Recv`** returns a `MPI_Status` type parameter that we haven't used so far. It is a structure integrated by 3 elements: `MPI_SOURCE`, `MPI_TAG` & `MPI_ERROR`. The first one contains the Rank of the sender process. If the message was received under `MPI_ANY_SOURCE` it can be necessary to find out who sent it later on in the program. The second one returns the message's tag. If it was received under `MPI_ANY_TAG`, it could be interesting to get to know the tag's value as well. The third one returns an error code. We won't deal with error codes in this exercise.

PRACTICAL EXERCISE

We will program a parallel matrix multiply. It is the student's decision how to scatter calculations among all the processes. The size of the matrices (square) must be configurable. Dynamic memory allocation is strongly recommended so no limits to the size of the matrices are imposed.

Process 0 will initialize the operand matrices with any value (random, loop, etc). Data type will be float. In a first stage, multiplication results will be displayed to check correctness. Once the program has been validated, result printing must be removed to allow matrix size to grow. Execution time has to be displayed in all cases.

REMARK:

To combine double indexing with dynamic memory allocation for matrices, we must use double pointers. Each pointer within an array will give access to a row in a matrix:

```c
// Declare a double poiter for the matrix
// This will let us refer to the elements in a [row][column] manner
float **Matrix;
// Initialize the double poiter to store poiters to each and every row in the matrix.
Matrix = (float **) malloc(ROWS*sizeof(float *));
// We initialize each poiter to the starting poit of each row
for (i=0; i< ROWS; i++)
{
        MatriX[i] = (float *) malloc(COLUMNS*sizeof(float));
}
// Now we can us [row][column] format for our matrix:
```

```
for (int i=0; i<ROWS; i++)
{
        for (int j=0; j<COLUMNS; j++)
        {
                Matrix[i][j] = 0.0;
        }
}
```

However, this dynamic allocation procedure does not guarantee that rows in the matrix are contiguous in memory. This can be necessary for sending functions in our program. We should send data row by row in that scenario. If we want to keep double indexing while adding contiguity, we will have to proceed as follows:

```
// Declare a double poiter for the matrix
// This will let us refer to the elements in a [row][column] manner
float **Matrix;
// Initialize the double poiter to store poiters to each and every row in the matrix.
MatriX = (float **) malloc(ROWS*sizeof(float *));
// Declare a new pointer to allocate memory space for the whole matrix.
float *Mf;
// Initialize the pointer that will guarantee consecutive location of all rows
Mf = (float *) malloc(ROWS*COLUMNS*sizeof(float));
// We initialize each poiter to the starting poit of each row.
for (i=0; i< ROWS; i++)
{
        MatriX[i] = Mf + i* COLUMNS;
}
// Now we can us [row][column] format for our matrix:
for (int i=0; i<ROWS; i++)
{
        for (int j=0; j<COLUMNS; j++)
        {
                Matrix[i][j] = 0.0;
        }
}
```

It is now important to notice that this alternative leads to the use of `Matrix[0]` as the starting address of the data stored in the matrix.

QUESTIONS

- In order to multiply A×B matrix A can be delivered to all processes whilst matrix B se can be distributed in columns. Think of a different option.
- Would it be possible to avail of the power of Cartesian topology to facilitate the resolution of this exercise?
- The need to broadcast one of the matrices slows program execution. Think of a different solution to avoid delivering so much information. Try to guess what the performance of this new option would be compared with the current program.

Activity 2: Performance Assessment

OBJECTIVES

- To measure system's performance in various circumstances.
- To learn how to estimate system's power and how to exploit it. A compromise between learning effort and code optimization must be obtained.

THEORETICAL CONCEPTS

In this chapter some common performance related concepts are presented:

- **Degree of parallelism (DOP)***:* Number of processors used to run a program in a precise moment on time. The curve, **DOP** = *P(t)*, representing the degree of parallelism as a function of time is called **parallelism profile** of the program. It doesn't need to match the number of processors available (*n*). For the following definitions we will assume that there are more processors than necessary to reach the maximum degree of parallelism admitted by a program: **máx**{*P(t)*} = *m* < *n*.

- **Total amount of work**: Being Δ the computation capacity of a single processor, given either in MIPS or MFLOPS, and assuming all processors to be equal, it is possible to measure the amount of work carried out between time instant t_A and t_B from the area under the parallelism profile as:

$$W = \Delta \cdot \int_{t_A}^{t_B} P(t) \cdot d \ .$$

  Usually the parallelism profile is a discrete graph (figure 3), so the total amount of work can be computed as:

$$W = \Delta \cdot \sum_{i=1}^{m} i \cdot t_i \, .$$

  Where t_i is the time span when the degree of parallelism is *i*, being *m* the maximum degree of parallelism all over the program's execution time.

  According to this, the sum of the different time intervals is equal to the program's execution time:

$$\sum_{i=1}^{m} t_i = t_B - t_A \, .$$

- **Average parallelism**: Is the arithmetic mean of the degree of parallelism along time:

$$\overline{P} = \frac{1}{t_B - t_A}\int_{t_A}^{t_B} P(t)\cdot dt = \frac{\sum_{i=1}^{m} i\cdot t_i}{\sum_{i=1}^{m} t_i}.$$

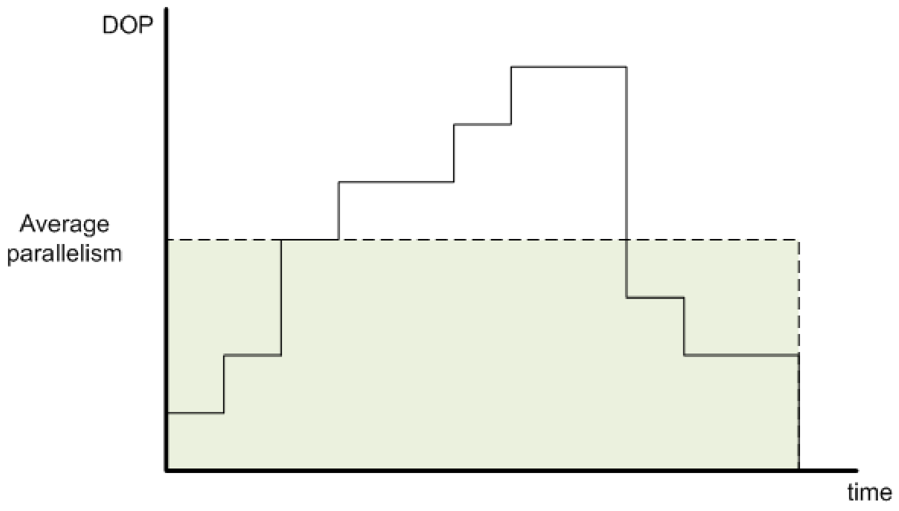


Figure 3.Parallelism profile and average parallelism.

- **Available parallelism**: Maximum degree of parallelism that can be extracted from a program, regardless of hardware constraints.

- **Asymptotic speedup**: Let $W_i = i\cdot\Delta\cdot t_i$ be the work done when **DOP** = *i*, hence $W = \sum_{i=1}^{m} W_i$.

  In this situation, the time employed by a single processor to carry out the work W_i is $t_i(1) = \frac{W_i}{\Delta}$; for *k* processors it is $t_i(k) = \frac{W_i}{k\cdot\Delta}$, and for an infinite number of processors it is $t_i(\infty) = \frac{W_i}{i\cdot\Delta}$.

  Hence, the **response time** is defined as:

$$T(1) = \sum_{i=1}^{m} t_i(1) = \sum_{i=1}^{m} \frac{W_i}{\Delta}$$

$$T(\infty) = \sum_{i=1}^{m} t_i(\infty) = \sum_{i=1}^{m} \frac{W_i}{i\cdot\Delta}.$$

  The **maximum speed-up** on a parallel system is reached when the number of processors is unlimited. It will be determined by the quotient of both:

$$S_{\infty} = \frac{T(1)}{T(\infty)} = \frac{\sum_{i=1}^{m} \frac{W_i}{\Delta}}{\sum_{i=1}^{m} \frac{W_i}{i \cdot \Delta}} = \frac{\sum_{i=1}^{m} \frac{i \cdot \Delta \cdot t_i}{\Delta}}{\sum_{i=1}^{m} \frac{i \cdot \Delta \cdot t_i}{i \cdot \Delta}} = \frac{\sum_{i=1}^{m} i \cdot t_i}{\sum_{i=1}^{m} t_i} = \overline{P}$$

It can be stated that the maximum speed-up for a parallel system with an unlimited number of processors is equal to the intrinsic average parallelism of the program to be parallelized. Obviously what is difficult is to figure out this intrinsic parallelism and make the program be as parallel as that.

A different way to calculate speed-up assumes that a job (being it either a single program or a group of them), is to be run in "*i*" mode if "*i*" processors are to be employed. In this scenario, R_i represents the collective speed of them all in either MIPS or MFLOPS; R_1 would be the speed of a single processor and $T_1 = 1/R_1$ the execution time. Let's suppose the job is conducted in "*n*" different modes, with different workload for each one, which results in a different weight f_i assigned to each mode. In this scenario, speed-up is defined as:

$$S = \frac{T_1}{T^*} = \frac{1/R_1}{\sum_{i=1}^{n} f_i/R_i}$$

Where T^* is the weighted harmonic mean of the execution time for the "*n*" execution modes.

In an ideal scenario, no delays are introduced by communications or lack of resources, so $R_1 = 1$, $R_i = i$:

$$S = \frac{1}{\sum_{i=1}^{n} f_i/i}$$

This expression is equivalent to the previous one.

From the previous case, the Amdahl's law is derived. $R_i = i$ and it is assumed that $W_1 = \alpha$ and $W_n = 1 - \alpha$, which implies that part of the work is to be done in sequential mode and the rest will exploit all system power. In this scenario:

$$S_n = \frac{1}{\frac{\alpha}{1} + \frac{1-\alpha}{n}} = \frac{n}{1 + (n-1)\alpha}$$

Hence:

$$n \to \infty \Rightarrow S \to 1/\alpha$$

In other words, system performance is upper bounded by the sequential part of the job.

- **System efficiency**: Determines the degree of exploitation of the resources available:

$$E = \frac{S}{n} = \frac{T(1)}{n \cdot T(n)} \leq 1$$

- **Redundancy**: Is the ratio between the number of operations performed by the system and those performed by a single processor to carry out the same job:

$$R = \frac{O(n)}{O(1)}$$

- **System utilization**:

$$U = R \cdot E = \frac{O(n)}{n \cdot T(n)}$$

- **Quality of parallelism**:

$$Q = \frac{S \cdot E}{R} = \frac{T^3(1)}{n \cdot T^2(n) \cdot O(n)}$$ assuming T(1) = O(1).

PRACTICAL EXERCISE

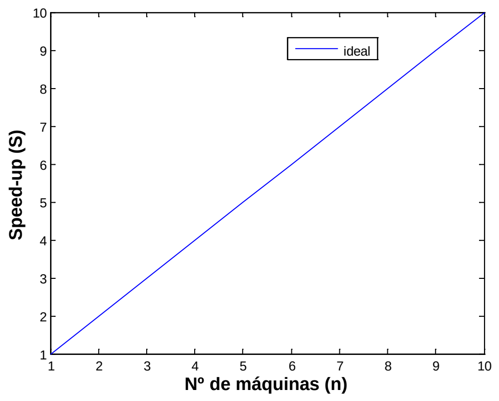
The program developed in the previous exercise (matrix multiply) is to be used as a benchmark to measure system performance. Matrix multiply is a cubic order problem that involves a significant calculation increase for a small increase in matrix size. In this exercise we will explore the influence of both system size and computation on execution time.

Concerning the amount of calculation, we must choose some precise values for matrix size. The first figure is intended to result in a similar execution time regardless of the amount of resources available. It will depend on the capabilities of the computers available. In our case we will start from matrices 3000×3000 in size. This leads to 27x10^9 multiplication operations.

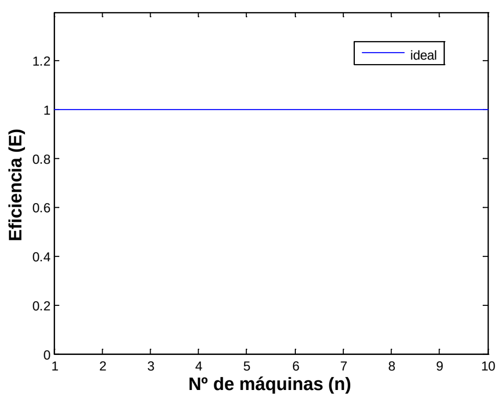
Starting from this size, we will increase matrix size to 4000 and 5000. For each of this values we will launch from 1 (2 in case process 0 doesn't perform calculations) to 6 (7) processes to be executed on the same number of computers. A graph representing execution time as a function of the number of computers (processes) should demonstrate that, when the workload is high, execution time is reduced proportionally to the number of resources deployed.

Speed-up graphs

Make a graph of the evolution of speed-up (*S*) and efficiency (*E*) as a function of the number of Computers and compare it with the ideal scenarios (figure a and figure b, respectively).



(a)



(b)

QUESTIONS

- For our experiments, determine: efficiency, utilization, redundancy and system quality.
- Compare the speed-up obtained with the one that should be achieved according to the amount of resources utilized.
- Try to figure out the reasons for the deviation.
- Describe which aspects should be improved to obtain a higher speed-up.

Activity 3: Introduction to Hybrid Programming

OBJECTIVES

- Understand the limitations of single thread parallel programming.
- Understand the benefits of multithread parallel programming.
- Learn how to embed memory sharing threads within distributed memory processes.

THEORETICAL CONCEPTS

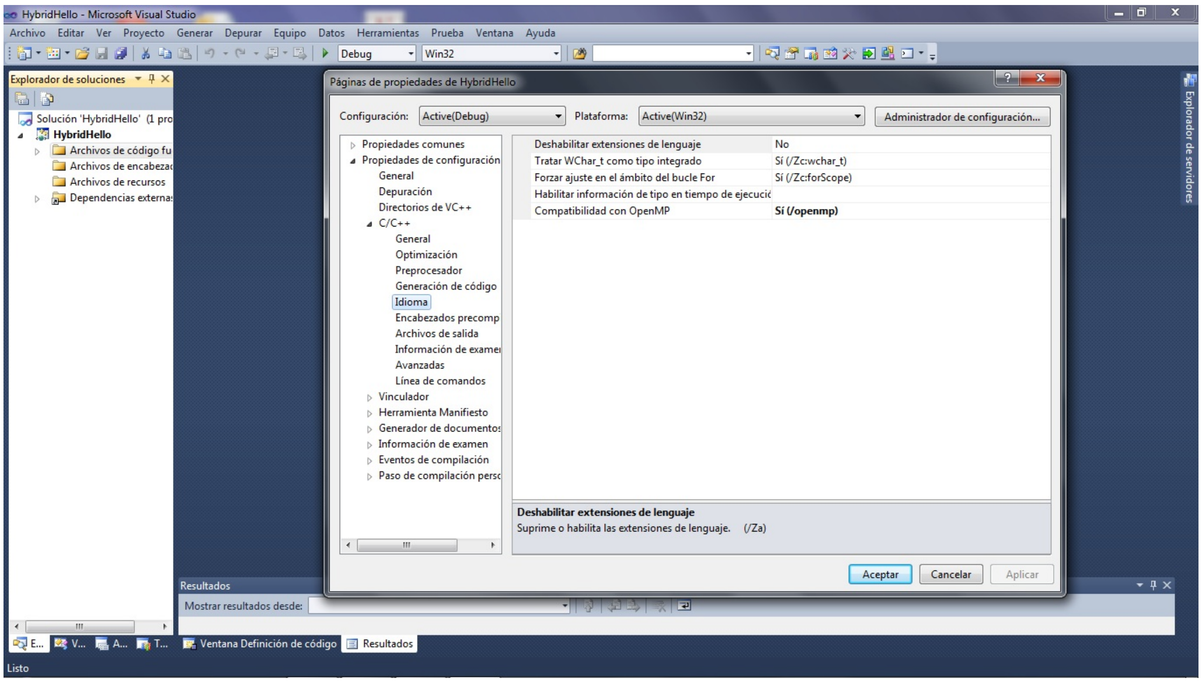
Limitations of single threaded parallel programs.

So far we have implemented single threaded application programs. There were several threads launched but only one per process. This has noticeable limitations when the program is intended to run in multicore systems which are the most common nowadays. If we launch as many processes as processors in the system, only one core on each processor will be busy. Efficiency can be improved simply by launching as many processes as cores. In that case a slightly more profound discussion will prove that there is still some inefficient use of resources to be fixed.

In message passing environments, such as MPI, information is exchanged by means of messages made up from the information itself and many other pieces of information: source, destination, tag, count, etc. When source and destination processes are physically located on different processor only connected by the local or system area network, that's fine and is actually the only way to share information. However, when processes are located on the same processor, they don't need to go over such a complicated procedure since they share a common memory where they both can read and write. In this scenario, the use of message introduces an overwhelming overhead. A shared memory environment must be put in place to improve efficiency to the top.

How to embed sharing memory threads into message passing processes.

So far we have implemented single threaded application programs. Each MPI process is single threaded. We can create multiple threads in different ways but we will choose the OpenMP programming environment for the sake of simplicity. To move to multithread programming, project settings must be changed:


HybridHello - Microsoft Visual Studio
Archivo Editar Ver Proyecto Generar Depurar Equipo Datos Herramientas Prueba Ventana Ayuda
Explorador de soluciones
Páginas de propiedades de HybridHello
Propiedades comunes
Propiedades de configuración
General
Depuración
Directorios de VC++
C/C++
General
Optimización
Preprocesador
Generación de código
Idioma
Aceptar
Cancelar
Aplicar

Configuración: Active(Debug) Plataforma: Active(Win32)

Deshabilitar extensiones de lenguaje	No
Tratar WChar_t como tipo integrado	Sí (/Zc:wchar_t)
Forzar ajuste en el ámbito del bucle For	Sí (/Zc:forScope)
Habilitar información de tipo en tiempo de ejecució	
Compatibilidad con OpenMP	**Sí (/openmp)**

Deshabilitar extensiones de lenguaje
Suprime o habilita las extensiones de lenguaje. (/Za)

Now the project is ready to accept OpenMP directives but some preliminary tips on this environment have to be provided.

OpenMP PARALLEL REGIONS

OpenMP programs are single threaded by default. They have a master thread that, in certain cases splits into a number of threads to perform a bunch of operations in parallel. There pieces of code are called “parallel regions”:

```
#pragma omp parallel
{
      Parallel region
}
```

Function omp_get_thread_num() returns current thread’s ID from 0 to N-1, but, how is N set? There are also many options here but in this course we will use only a static explicitly defined one:

```
#pragma omp parallel num_threads (N)
{
      Parallel region
}
```

This is fine but, we still need to define the contents of the parallel region inside. This could be integrated by any possible collection of valid instructions but in this programming environment parallel regions are used most of the times to parallelize loops. “For” loops are the ideal candidates:

```
#pragma omp parallel num_threads (N)
{
      #pragma omp for
```

```
            for(i=0;i<n;i++){
                  Operations to be performed
            }
}
```

The “n” operations to be performed will be scattered among the N threads. That will hopefully result in a reduction of execution time in case of multicore/multithreaded processors.
This is a shared memory environment but, where are the shared variables? Variables declared outside the parallel region are shared. Variables declared inside the parallel region are private to each thread. Still it is possible to turn a shared variable into a private one:

```
#pragma omp parallel num_threads (N) private (j)
{
      #pragma omp for
            for(i=0;i<n;i++){
                  Operations to be performed on variable j
            }
}
```

In this case, each thread will have its own copy of “j” even though it was declared outside the region but, what would be j’s value on each thread? In the previous piece of code “j” is not initialized regardless the value it might have before the region. If we want use its previous value to initialize each thread’s copy:

```
#pragma omp parallel num_threads (N) firstprivate (j)
{
      #pragma omp for
            for(i=0;i<n;i++){
                  Operations to be performed on variable j
            }
}
```

Likewise, we may need the master thread to be aware of the changes suffered by “j” inside the region once it finishes. We can force the value of “j” to be the last one taken inside the region:

```
#pragma omp parallel num_threads (N) firstprivate (j) lastprivate (j)
{
      #pragma omp for
            for(i=0;i<n;i++){
                  Operations to be performed on variable j
            }
}
```

To end up this brief introduction, we will have a look at an additional capability of OpenMP. It won’t be hard to understand since there is an equivalent one in MPI we have already used. This is the reduction operation. It applies to a situation where a shared variable is being modified into different values by different threads. Sometimes the final value of this variable has to be obtained from a combination of the values generated by the different threads. Let's have a look at the example:

```
#pragma omp parallel num_threads (N)
{
        #pragma omp for reduction(+:sum)
                for (i=0;i<n;i++){
                        sum=sum+(a[i]);
                }
}
```

It is obvious that we intend to obtain a final value of "sum" which should be the result of the "n" sums performed on it. The reduction clause will take the last value generated by each thread and then perform a final sum on all of them. To make this possible, a private copy of the shared variable is generated on each thread.

PRACTICAL EXERCISE

Take again the matrix multiply program and conduct the following experiments:

1. Perform the multiplication on 5000x5000 matrices. Launch two processes.
2. Do the same with as many processes as processor cores available.
3. Do it again with one more process than cores.
4. Now adapt your program to the hybrid programing paradigm launching two processes and splitting the working one in as many threads as cores minus one.
5. Run the same hybrid program with as many threads as cores.

Compare the time spent by the different experiments and answer the following

QUESTIONS

- Which programming paradigm provides de highest performance?
- Is it more optimal to run only one process/thread on each core or it turns out that process 0 must share core with another process/thread?
- Are these results what we could expect? Why?

Activity 4: Hybrid Programming

OBJECTIVES

❖ Understand some work scheduling options in order to optimize execution time.

THEORETICAL CONCEPTS

Synchronization.

The default synchronization procedure introduces a barrier at the end of the parallel region so execution does not continue until all threads reach that point. This is a sensible thing to do but, in certain cases, it may be useful to avoid that constraint. This can be done by means of the “nowait” clause.

```
#pragma omp parallel num_threads (N)
{
      #pragma omp for nowait
            for(i=0;i<n;i++){
                  Operations to be performed on variable j
            }
}
```

In this particular example it doesn’t make any difference but, in case we had another parallel loop right after, it would save time if some threads could enter it as soon as possible.

Scheduling.

So far we have assumed that the amount of work to be done is delivered to the different threads in a fair manner. That’s right but, even in this case there could be different possibilities that result in significant performance variations.

The default scheduling policy divides the number of iterations by the number of threads thus giving each thread the same amount of work if possible. This work is assigned prior execution and no changes are made at run time. It is possible to specify different work “chunks”. In this case each particular implementation decides how to allocate chunks on threads.

```
#pragma omp parallel num_threads (N)
{
      #pragma omp for schedule(static,10)

            for(i=0;i<n;i++){
                  Operations to be performed on variable j
            }
}
```

In this example chunks of 10 iterations are delivered. The last chunks are made smaller when necessary.

Static policies do not allow to dynamically assigning pieces of work to threads as they finish their previously assigned one. This results in a loss of efficiency that should be avoided. Dynamic policies can be applied to do so.

```
#pragma omp parallel num_threads (N)
{
      #pragma omp for schedule(dynamic,10)

            for(i=0;i<n;i++){
                  Operations to be performed on variable j
            }
}
```

In this example, threads get new chunks as soon as they finish their current calculation.

PRACTICAL EXERCISE

We will continue the experiments done on the previous exercise. We already have the results obtained from the default static scheduling. Now we will add these new ones:

- Try again the static scheduling but specifying a chunk size of 10.
- Then try chunk size 100.
- Now shift to dynamic scheduling with chunk size 10.
- Try again dynamic with chunk size 100.

Compare all results to see which the best policy is and try to explain why.
Work with the number of threads that proved to be the best option in the previous exercise.

REFERENCES:

OPENMP APPLICATION PROGRAM INTERFACE. Available at:
http://www.openmp.org/mp-documents/spec30.pdf

Activity 5: MPI vs OpenMP

OBJECTIVES

❖ In hybrid programming many different number of threads and processes may be launched. We will try to find out which is the best combination.

❖ Message passing and shared memory involve different programming techniques and a distinct use of hardware resources. We need to know which one is more efficient and then more convenient.

THEORETICAL CONCEPTS

No additional theoretical discussion will be introduced for this exercise.

PRACTICAL EXERCISE

We will launch a battery of test meant to fulfill the first of the objectives already stated:

- Repeat the matrix multiplication on two 5000x5000 matrices with two MPI processes in the local machine.
- Launch as many MPI processes as cores are available.
- Launch as many MPI processes as cores are available plus one.
- Back to two MPI processes split the working one (rank 1) into as many threads as cores available minus one.
- Split rank 1 into as many threads as cores are available so one of its threads will share a core with rank 0 process.

Compare all results to see which the best policy is and try to explain why.

Now we will address the second objective. Use the 5000 x 5000 case again:

- Launch as many processes as processors available plus one and split the working processes into as many threads as cores available.
- Launch as many processes as cores available plus one (no shared memory this time).

Compare the results and try to explain them. See references to find answers.

REFERENCES:
Comparing the OpenMP, MPI, and Hybrid Programming Paradigms on an SMP Cluster
Gabriele Jost and Haoqiang Jin and Dieter An Mey and Ferhat F. Hatay
NAS Technical Report NAS-03-019, November 2003.

Activity 6: Submitting jobs to a cluster

OBJECTIVES

- Get to know how Jobs are submitted to a computation cluster.
- Understand the differences between a local working environment and a cluster architecture

THEORETICAL CONCEPTS

The Jobs we are about to submit to the cluster are no different from those we have been working with so far. They will be MPI programs mainly derived from the matrix multiply application we are using as a benchmark. We will work in Windows 8.1 using the user roles previously generated within the ARAVAN workgroup and also within the HPC (High Performance Computing) cluster. The user will be allowed to launch jobs to the cluster. From now we are going to use the Microsoft MPI implementation: MS-MPI.

The tool used to submit these Jobs is the "Job Manager" and it is part of the client tools installed by the HPC PACK 2012 R2. Before we can send jobs to execution there are a few issues we have to deal with:

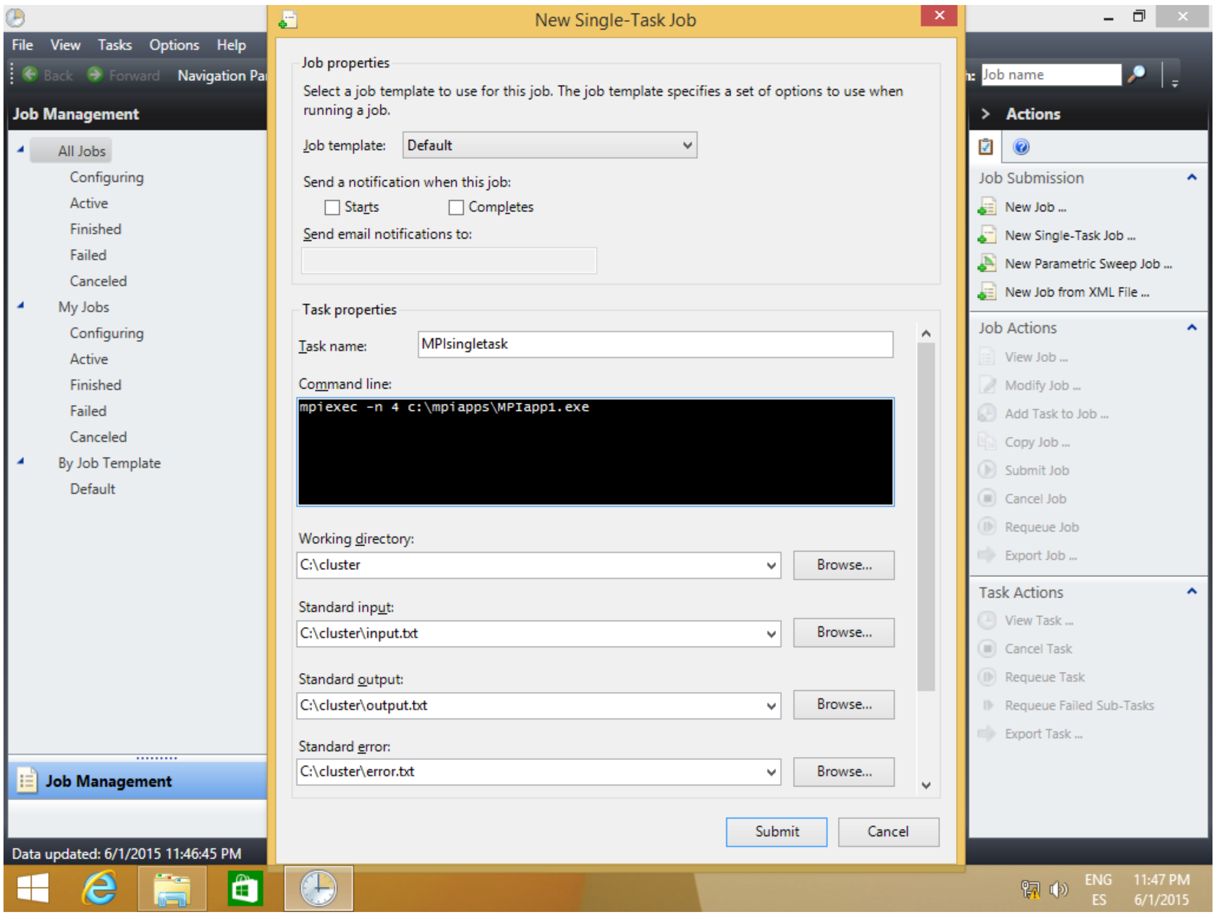
1. We won't have a GUI. Initialization information will be parsed to the applications from the command line. Other information needed at run time has to be provided within a file. Therefore it will be necessary to adapt our programs to these situations in certain cases. Concerning the matrix multiply program we have developed, matrix size will be introduced as an initialization parameter from command line. A code line simliar to: "size = atoi (argv[1]);" will provide the numerical value of this parameter so it can be used within the program.

2. Program's output will be redirected to a text file we will have to open once the program has finalized to see the results.

3. The job manager will consider the program's execution unsuccessful unless it returns a zero code. We can write "exit (0)" at the end of the program to do so. It is defined within <stdlib.h>.

Local job generation with Job Manger.

A job is integrated by a number of tasks. Tasks are user applications meant to be executed by the system. We mean to launch jobs comprising one single task: our MPI application. In this case we can use the option "Single Task Job" to make the process simple.



New Single-Task Job

Job properties

Select a job template to use for this job. The job template specifies a set of options to use when running a job.

Job template: Default

Send a notification when this job:
- [ ] Starts
- [ ] Completes

Send email notifications to:

Task properties

Task name: MPIsingletask

Command line:

```
mpiexec -n 4 c:\mpiapps\MPIapp1.exe
```

Working directory: C:\cluster

Standard input: C:\cluster\input.txt

Standard output: C:\cluster\output.txt

Standard error: C:\cluster\error.txt

Data updated: 6/1/2015 11:46:45 PM

ENG ES 11:47 PM 6/1/2015

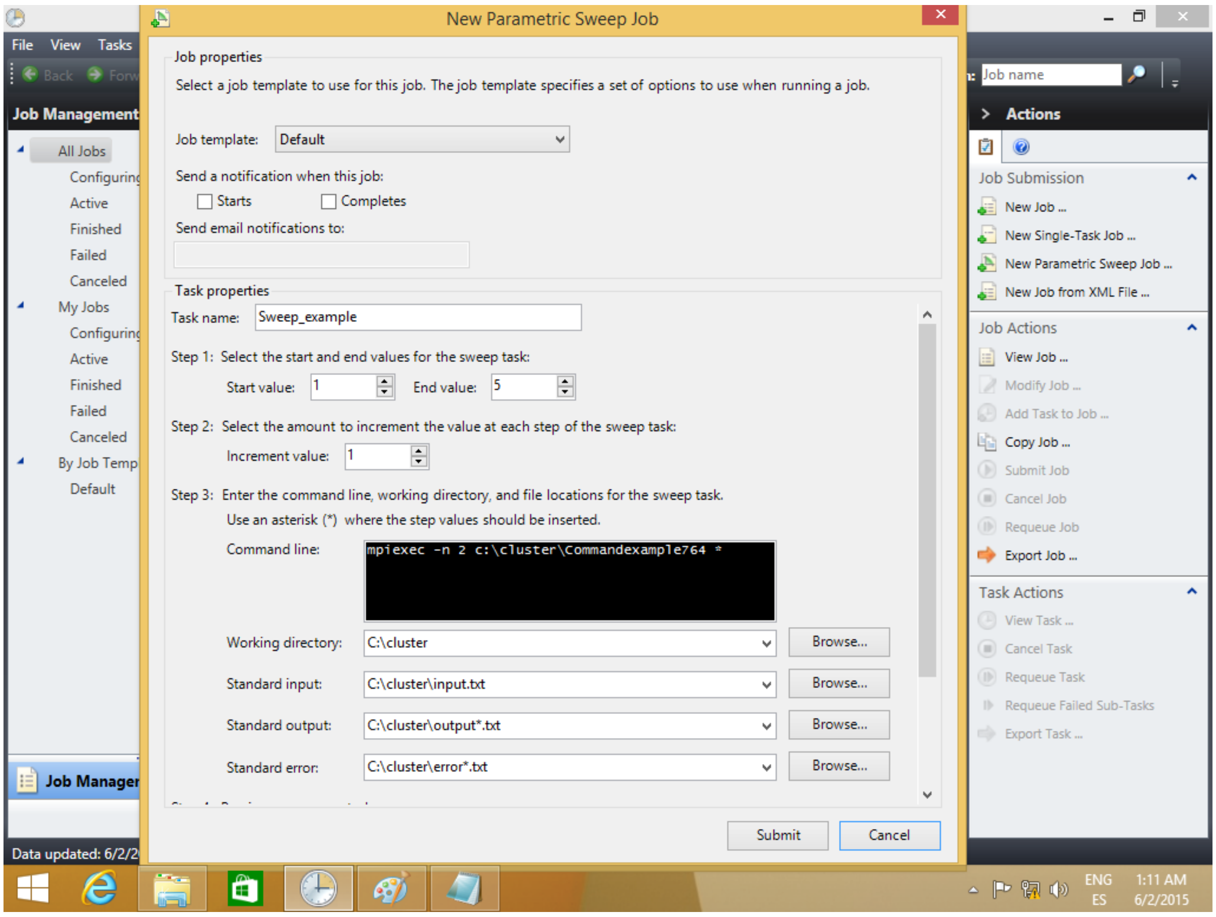
Figure 6.1. Single task job configuration.

We have to select the working directory. In this case we introduce the folder where the input and output text files are to be placed. We also introduce the names for these files. If no input data is required the “Standard input” field may be left blank.

On the command line we describe the task to be performed, a parallel MPI application in this case: “mpiexec –n 4 c:\mpiapps\MPIapp1.exe”. It doesn’t need to be located in the working directory. The “-n 4” parameter tells the system to launch 4 processes.

Parametric sweep jobs.

In many real situations, tasks are not performed individually but rather in a combined manner so results can be analyzed and compared. As a matter of fact, we usually launch many executions of our matrix multiply program to see how different configurations and sizes affect execution time. It is possible to launch a job for each case but it would be more efficient to launch them all together. This is what the “Parametric sweep job” option makes possible.



New Parametric Sweep Job

Job properties

Select a job template to use for this job. The job template specifies a set of options to use when running a job.

Job template: Default

Send a notification when this job:

☐ Starts ☐ Completes

Send email notifications to:

Task properties

Task name: Sweep_example

Step 1: Select the start and end values for the sweep task:

Start value: 1 End value: 5

Step 2: Select the amount to increment the value at each step of the sweep task:

Increment value: 1

Step 3: Enter the command line, working directory, and file locations for the sweep task.

Use an asterisk (*) where the step values should be inserted.

Command line:

```
mpiexec -n 2 c:\cluster\Commandexample764 *
```

Working directory: C:\cluster

Standard input: C:\cluster\input.txt

Standard output: C:\cluster\output*.txt

Standard error: C:\cluster\error*.txt

Figure 6.2. Parametric sweep job configuration.

In this example we have set the parameter to vary from 1 to 5 incrementing one by one. As a result, 5 tasks will be conducted, one for each of its values. The asterisk used to place the parameter in the command line is also placed within the names of the text files so each task is linked to its own output file.

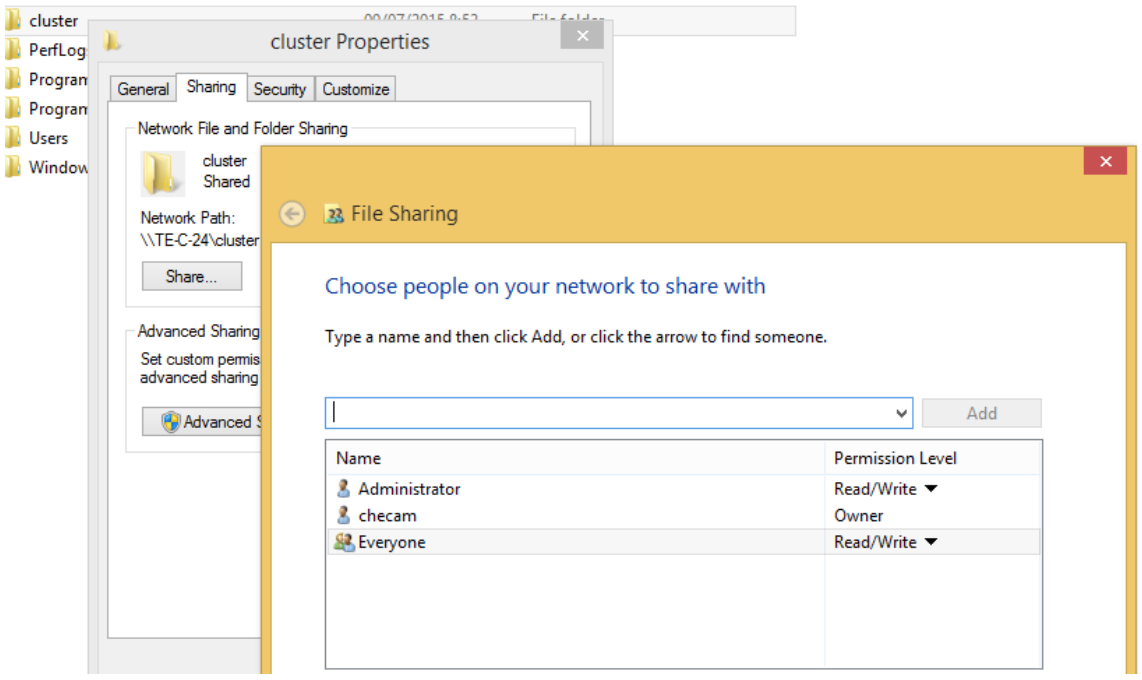
In this example we have used the parameter to modify the command line argument parsed to the program but it can connected with any other aspect of the information provided in the command line. For instance, we could vary the number of processes to be launched instead: “mpiexec –n * c:\ruta\multimatriz 5000”. We could provide more than one asterisk in the same command line but it is very unlikely that the same values make sense in different positions. Netting parameters within the same task is not permitted.

Job generation with Job Manger for the cluster.

Generating jobs for the local node or for the cluster is conceptually the same, since the former is just a section of the later. Nevertheless is important to remark in this section some settings to be made:

- Folder and subfolder sharing.
- Working directory configuration.
- Node selection.

For the applications to be executed by remote nodes, the working directory must be shared. We can use Windows Explorer to edit the properties of the folder containing the working directory and then share it.

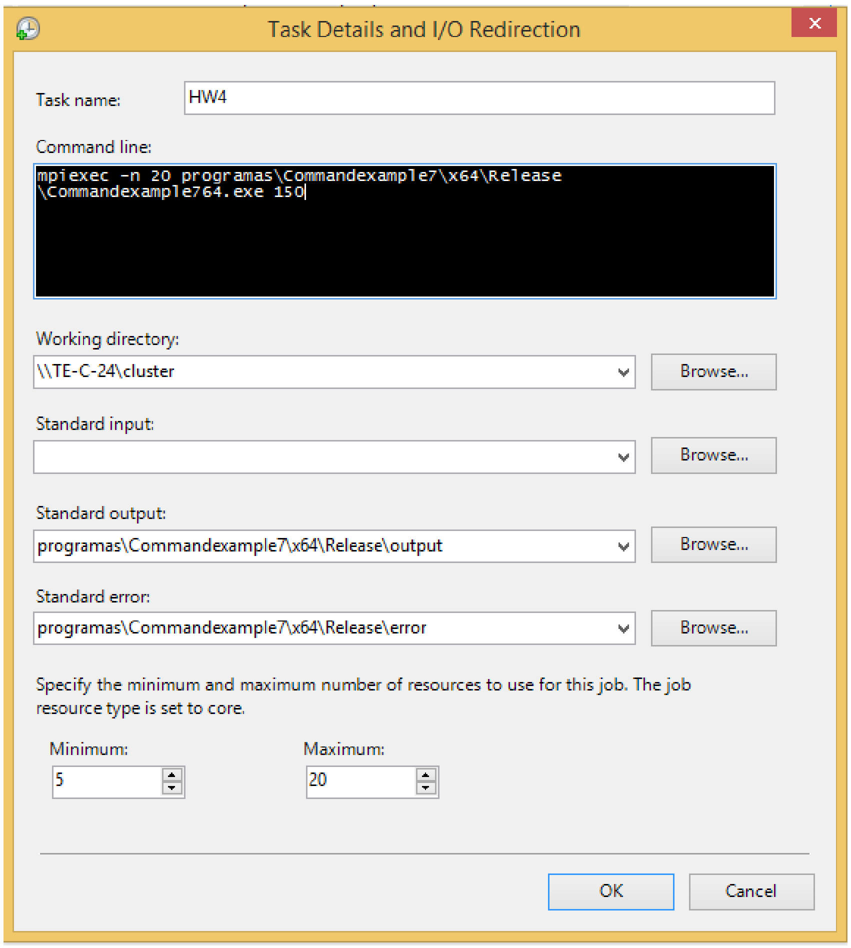


Name	Permission Level
Administrator	Read/Write
checam	Owner
Everyone	Read/Write

Figure 6.3. Sharing the working directory.

Users meant to execute the application must have the appropriate rights.

If the job is to be executed by other nodes, its path must be known under a common format. UNC (https://msdn.microsoft.com/en-us/library/gg465305.aspx) is the one accepted for this purpose. It is use to declare the path for the working directory. The rest of paths: input and output files and the application itself are referred to the working directory as a relative path. Figure 6.4 shows how to make these settings.



Task name: HW4

Command line:

```
mpiexec -n 20 programas\Commandexample7\x64\Release
\Commandexample764.exe 150
```

Working directory: \\TE-C-24\cluster

Standard input:

Standard output: programas\Commandexample7\x64\Release\output

Standard error: programas\Commandexample7\x64\Release\error

Specify the minimum and maximum number of resources to use for this job. The job resource type is set to core.

Minimum: 5

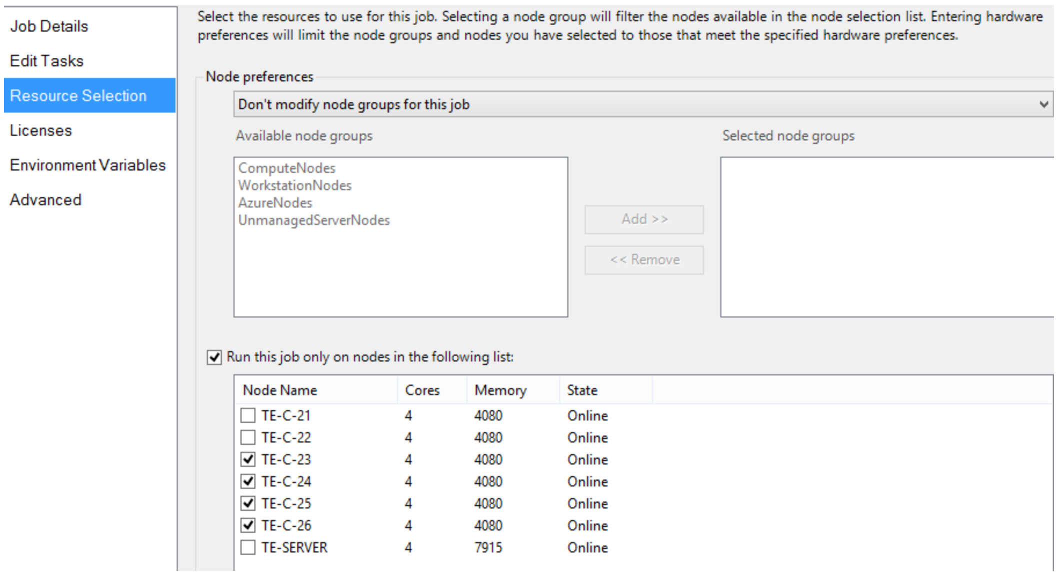
Maximum: 20

Figure 6.4. Configuratiopn of the shared working directory.

In this particular case the application's whole path would be:

TE-C-24\c:\cluster\programas\Commandexample7\x64\Release\Commandexample764.exe, where "150" is a command line argument for the application.

When configuring a new job, the "Resource Selection" option will display the available nodes on the cluster so we can select the desired ones.



Select the resources to use for this job. Selecting a node group will filter the nodes available in the node selection list. Entering hardware preferences will limit the node groups and nodes you have selected to those that meet the specified hardware preferences.

Node preferences

Don't modify node groups for this job

Available node groups: ComputeNodes, WorkstationNodes, AzureNodes, UnmanagedServerNodes

Selected node groups

☑ Run this job only on nodes in the following list:

	Node Name	Cores	Memory	State
☐	TE-C-21	4	4080	Online
☐	TE-C-22	4	4080	Online
☑	TE-C-23	4	4080	Online
☑	TE-C-24	4	4080	Online
☑	TE-C-25	4	4080	Online
☑	TE-C-26	4	4080	Online
☐	TE-SERVER	4	7915	Online

Figure 6.5. Node selection.

PRACTICAL EXERCISE

First we will launch the classic HelloWorld application. It will display the usual waiving message along the process rank and the total number of processes.

The, going back to the matrix multiply program, it has to be adapted to the new situation. Matrix size has to be entered from the command line as already explained. We will launch two parametric jobs: one of them will vary the number of processes from 2 to 8 and the other will vary matrix size from 3000 to 5000 in steps of 1000.

- Copy all outputs in the activity report.
- The report must include the resources assigned to the different tasks. This can be obtained from the task report provided once it has been finalized.
- Check that the time the system declares to have invested in each task matches the one provided by the matrix multiply as its output result.

Activity 7: Job scheduling

OBJECTIVES

- ❖ Understand job scheduling policies.
- ❖ Analyze their impact on overall system performance.

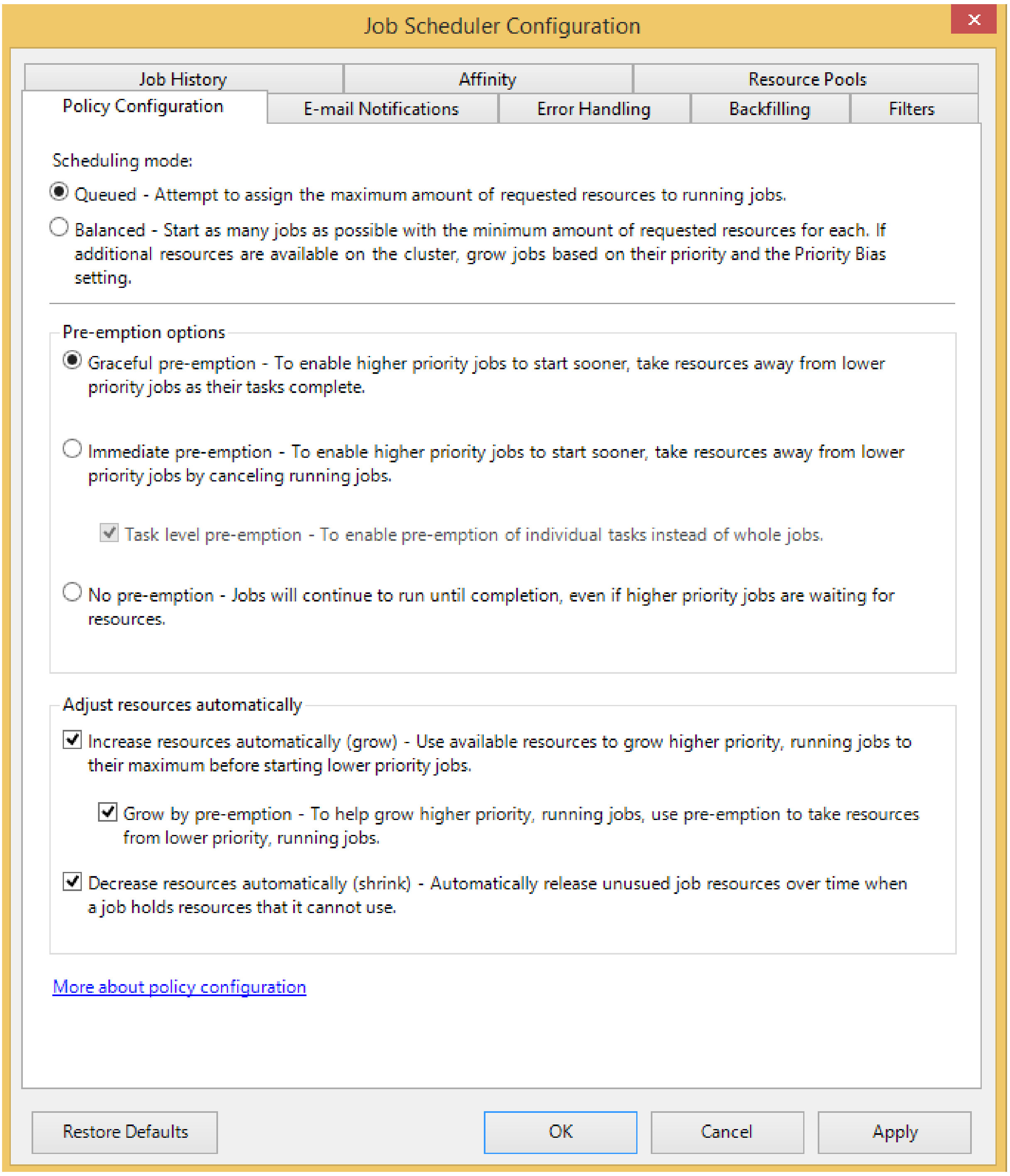
THEORETICAL CONCEPTS

The system administrator is responsible for the establishment of efficient scheduling policies. The goal is to optimize system performance. What this means is not obvious though. There may be several ways of interpreting performance and therefore different objectives to meet:

- Maximize system utilization.
- Minimize job execution time (wall time).
- Maximize throughput (jobs done per time unit).

From the user's prospective, wall time is usually what matters but, the administrator is not expected to serve one privileged user but rather to make the most of the system as a whole.
Windows HPC Job Scheduler provides several scheduling options. The first choice to be made is whether to launch jobs in a queued or in a balanced way:

- Queued: the scheduler will try to give the maximum requested resources to incoming jobs and to incoming tasks within a job. When all resources are exhausted, next jobs will have to queue up. As shown in the figure, several sub options accompany this decision.



Job Scheduler Configuration

Scheduling mode:

- [✓] Queued - Attempt to assign the maximum amount of requested resources to running jobs.
- [ ] Balanced - Start as many jobs as possible with the minimum amount of requested resources for each. If additional resources are available on the cluster, grow jobs based on their priority and the Priority Bias setting.

Pre-emption options

- [✓] Graceful pre-emption - To enable higher priority jobs to start sooner, take resources away from lower priority jobs as their tasks complete.
- [ ] Immediate pre-emption - To enable higher priority jobs to start sooner, take resources away from lower priority jobs by canceling running jobs.
  - [✓] Task level pre-emption - To enable pre-emption of individual tasks instead of whole jobs.
- [ ] No pre-emption - Jobs will continue to run until completion, even if higher priority jobs are waiting for resources.

Adjust resources automatically

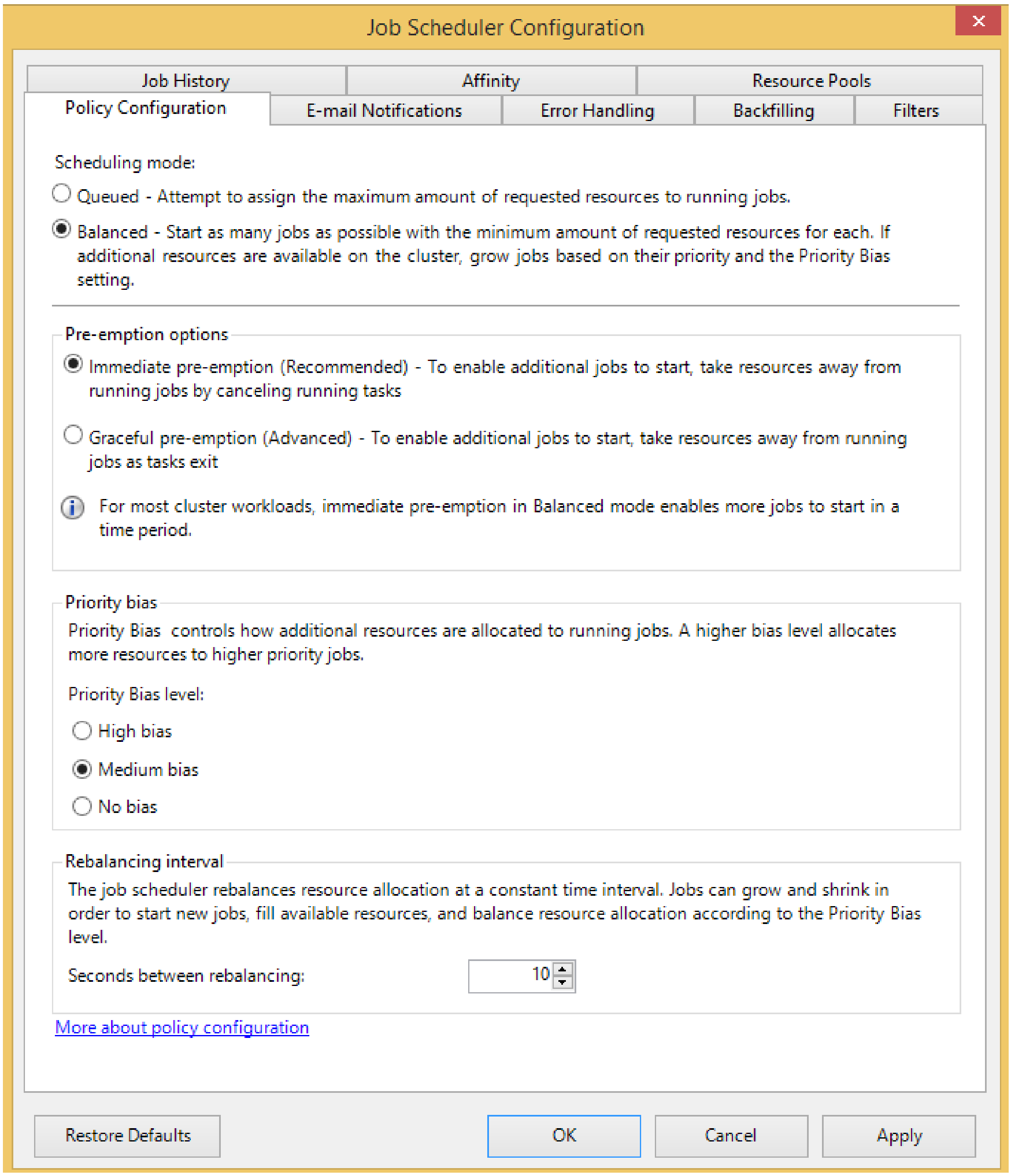
- [✓] Increase resources automatically (grow) - Use available resources to grow higher priority, running jobs to their maximum before starting lower priority jobs.
  - [✓] Grow by pre-emption - To help grow higher priority, running jobs, use pre-emption to take resources from lower priority, running jobs.
- [✓] Decrease resources automatically (shrink) - Automatically release unusued job resources over time when a job holds resources that it cannot use.

More about policy configuration

Figure 7.1. Queued scheduling policy.

One of them is how to deal with preemption. It can be graceful so higher priority jobs may take resources away from others but only when individual tasks within them finish. It can be immediate so lower priority jobs are cancelled in order to serve higher priority ones. Or it can be disabled. Resources may be set to adjust automatically. Higher priority jobs may be given more resources until they reach the maximum requested before any lower priority job is launched. They can even grow taking away resources from already running lower priority jobs. Finally resources may be set to be taken away from jobs that no longer will make use of them.

- Balanced: the scheduler will try to start as many jobs as possible. To do so it will reduce de amount of resources allocated to each one but not below the minimum requested by the user. Therefore, as long as there are resources available, new jobs will be launched rather than increasing the resources to already running jobs. As in the previous case, a number of sub-options become available.



Scheduling mode:

( ) Queued - Attempt to assign the maximum amount of requested resources to running jobs.

[✓ Balanced] - Start as many jobs as possible with the minimum amount of requested resources for each. If additional resources are available on the cluster, grow jobs based on their priority and the Priority Bias setting.

Pre-emption options

[✓ Immediate pre-emption (Recommended)] - To enable additional jobs to start, take resources away from running jobs by canceling running tasks

( ) Graceful pre-emption (Advanced) - To enable additional jobs to start, take resources away from running jobs as tasks exit

For most cluster workloads, immediate pre-emption in Balanced mode enables more jobs to start in a time period.

Priority bias

Priority Bias controls how additional resources are allocated to running jobs. A higher bias level allocates more resources to higher priority jobs.

Priority Bias level:

( ) High bias

[✓ Medium bias]

( ) No bias

Rebalancing interval

The job scheduler rebalances resource allocation at a constant time interval. Jobs can grow and shrink in order to start new jobs, fill available resources, and balance resource allocation according to the Priority Bias level.

Seconds between rebalancing: 10

Figure 7.2. Balanced scheduling policy.

Yet again, pre-emption can be selected as either graceful or immediate but this time it is not possible to disable it. When it comes to allocating additional resources to running jobs, the decision of how to do it is biased by priority. How big this influence is can be adjusted. Finally, the time elapsed between rebalancing decisions can be set as well.

Priority is assigned by the user to a job at its configuration. Individual tasks within a job share the jobs priority. In this activity we will work with single multitasked jobs, so we will ignore all what has to do with priority for the moment.

PRACTICAL EXERCISE

We are going to set up a job integrated by a number of tasks. Each task will consist of the execution of a 8000 x 8000 matrix multiplication. The number of processes will increase from 4 to 4 multiplied by the number of computers available in the cluster so the last task may potentially make use of all cores in the system. The number of processes will increase in 4 over the previous taks thus determining the total number of tasks to be in the job.

The job will be launched under queued policy and then under balanced policy. On each case the tasks will be sorted according to the number of processes launched. It will be done from lowest to highest and then from highest to lowest. Four tests will then be conducted in total.

For each case build up a table to show the resources allocated to each task, its execution time and the overall execution time.

What scheduling policy turns out to be the best for this type of load? Using the contents of the tables, try to explain why.

Activity 8: Job scheduling II

OBJECTIVES

- ❖ Understand the differences between tasks and jobs.
- ❖ Working with different priority levels and scheduling policies.

THEORETICAL CONCEPTS

In this activity we will work with multiple jobs so priority configuration will become available to us. Different jobs may be assigned different priority levels either because their importance to the user is different or in order to optimize performance. In this exercise we will try to do the latest.

Priority levels can be set for each job as:

- Highest.
- Above normal.
- Normal.
- Below normal.
- Lowest.

When trying to work with jobs a new issue will probably arise. The top number of simultaneous connections to the share folder (up to 20 are permitted by the operating system) may be exceeded. The solution to this problem is to work with shared folders located in the server, whose operating system allows an almost unlimited number of connections. Set the folder provided by the system administrator as working directory, copy your .exe file there and everything should work fine.

PRACTICAL EXERCISE

First, we will repeat the experiments conducted in the previous exercise but launching multiple single task jobs instead of one multitasked job in each case. All jobs will keep their default normal priority level for the moment. Rebuild the tables generated in the previous activity with the new results and compare both.

Are execution times better or worse this time? What other circumstances arise now? How could them be overcome?

Now, keeping the scheduling settings as default, change priority levels to what you expect to be the best for your experiments and do the same as before.

Have you managed to improve performance? Why do you think is that?

Activity 9: Job scheduling III

OBJETIVES

- Understanding preemption.
- Checking the influence of preemption on system performance.

THEORETICAL CONCEPTS

Preemption allows higher priority jobs to interrupt lower priority ones. As shown before, this can be done in different ways. Since our goal remains system performance, higher priority should be given so the overall execution time is minimized.

PRACTICAL EXERCISE

Group the tasks launched in previous scheduling activities in two jobs. On one job the tasks comprising less processes will be placed and this job will be given the highest priority. The other job, with the lowest priority will entail the rest of the tasks.

Under both queued and balanced scheduling policies, repeat the usual experiments trying the different preemption options available.

Build up again the tables and compare results.

Decide what preemption policy is the most advisable for this type of workload.

Compare the results obtained under queued scheduling policy with and without the clicks on the "Adjust resources automatically" options.

Compare the results obtained under balanced scheduling policy using the different biasing options available.

Activity 10: Performance competition.

OBJECTIVES

- Making the best scheduling decisions.

THEORETICAL CONCEPTS

No theoretical concepts are introduced in this activity.

PRACTICAL EXERCISE

For a given matrix multiplication application (the same for all participants), each one will make what are expected to be the best scheduling decisions. This will include using jobs, tasks or both. Once they are made, the usual experiments will be conducted and the overall execution times compared in order to find out what were actually the best scheduling options.

In your report include your decisions, your results and compare them with the best performer. Explain why you think your decisions were not the best. If you are the best performer, congratulations, you will save some work.

Appendix A: Installing DeinoMPI

DeinoMPI in an implementation of the standard MPI-2 for Microsoft Windows derived from Argonne Nacional Laboratory's MPICH2.

System requirements:

- Windows 2000/XP/Server 2003/Windows 7
- .NET Framework 2.0

Installation

DeinoMPI has to be downloaded and then installed in all computers in the cluster. The installation process is the same in all nodes. It requires administrator privileges for installation but all users can execute it afterwards.

Once it is installed folder \bin has to be added to the path.

Note: make sure Deino's version matches the operating systems requirements (32 or 64 bits).

Configuration

Once the software has been installed, each user will need to create a "Credential Store". It is used to launch routines in a secure manner. Mpiexec will not execute any of them without this "Credential Store". The graphic environment will show the user this option in the first execution.

Launching Jobs

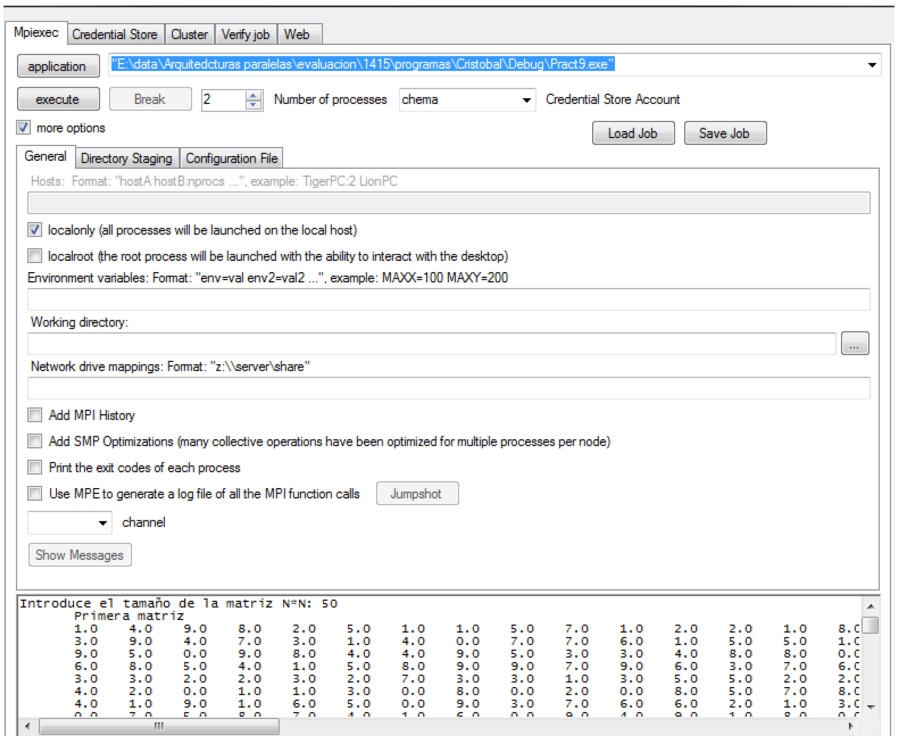
Once again, both the graphic environment and the command line are valid.

Graphic Environment

This tool can be used to launch MPI processes, manage the "Credential Store", search for computers within the local network that have MPI installed, verify mpiexec entries to diagnose common problems, and go to the DeinoMPI web site to look for help and documentation.

Mpiexec tab

It is the main page and is used to launch and manage MPI processes.



"E:\data\Arquitedcturas paralelas\evaluacion\1415\programas\Cristobal\Debug\Pract9.exe"

Number of processes: 2

Credential Store Account: chema

☑ more options

Hosts: Format: "hostA hostB:nprocs ...", example: TigerPC:2 LionPC

☑ localonly (all processes will be launched on the local host)

☐ localroot (the root process will be launched with the ability to interact with the desktop)

Environment variables: Format: "env=val env2=val2 ...", example: MAXX=100 MAXY=200

Working directory:

Network drive mappings: Format: "z:\\server\share"

☐ Add MPI History

☐ Add SMP Optimizations (many collective operations have been optimized for multiple processes per node)

☐ Print the exit codes of each process

☐ Use MPE to generate a log file of all the MPI function calls

channel

```
Introduce el tamaño de la matriz N*N: 50
        Primera matriz
        1.0     4.0     9.0     8.0     2.0     5.0     1.0     1.0     5.0     7.0     1.0     2.0     2.0     1.0     8.
        3.0     9.0     4.0     7.0     3.0     1.0     4.0     0.0     7.0     7.0     6.0     1.0     5.0     5.0     1.
        9.0     5.0     0.0     9.0     8.0     4.0     4.0     9.0     5.0     3.0     3.0     4.0     8.0     8.0     0.
        6.0     8.0     5.0     4.0     1.0     5.0     8.0     9.0     9.0     7.0     9.0     6.0     3.0     7.0     6.
        3.0     3.0     2.0     2.0     3.0     2.0     7.0     3.0     3.0     1.0     3.0     5.0     5.0     2.0     2.
        4.0     2.0     0.0     1.0     1.0     3.0     0.0     8.0     0.0     2.0     0.0     8.0     5.0     7.0     8.
        4.0     1.0     9.0     1.0     6.0     5.0     0.0     9.0     3.0     7.0     6.0     6.0     2.0     1.0     3.
```

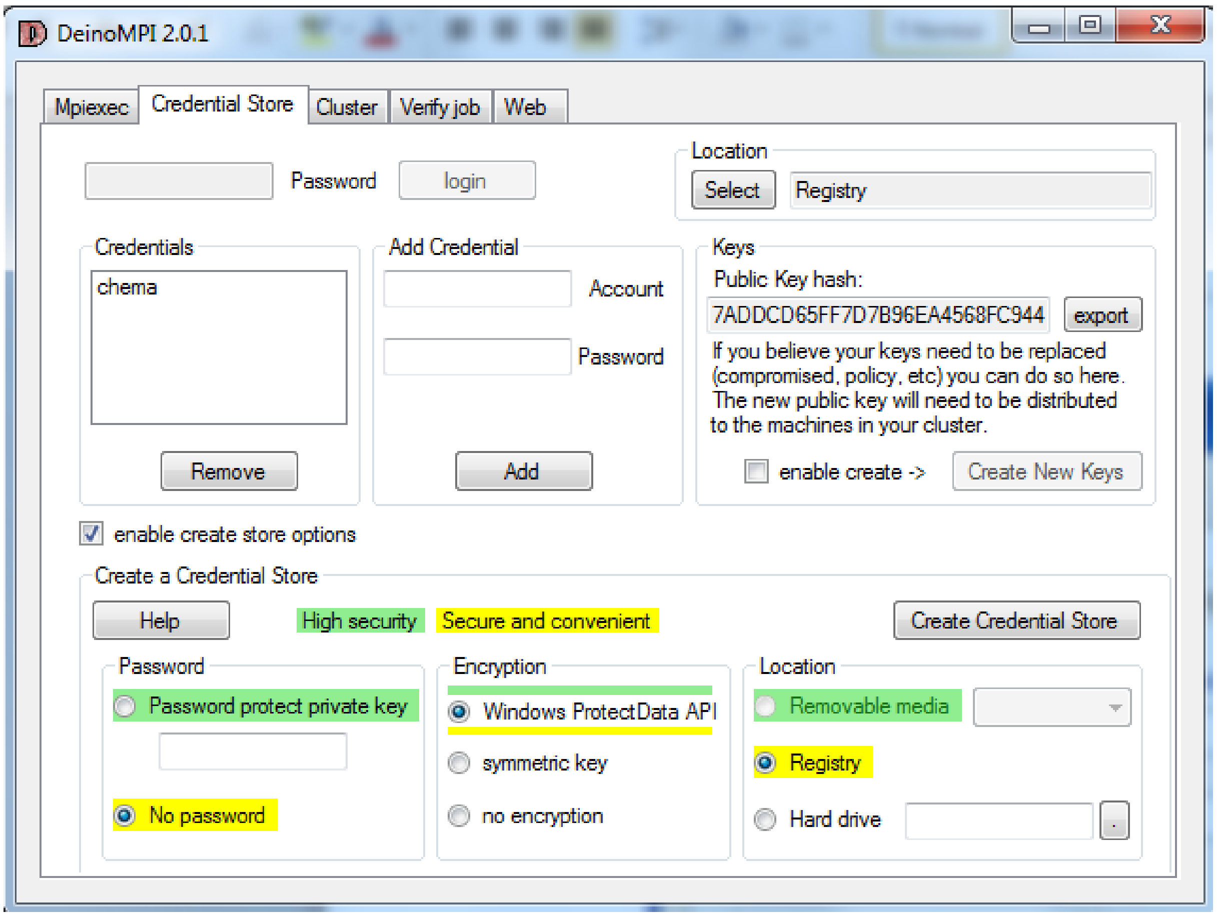
Figure A1. Mpiexec tab.

These are the main elements of this tab:

- **Application:**
  - The MPI application's path is introduced here. The same path will be taken by default in all nodes within the cluster so it is recommendable to copy the .exe file in the same folder in all of them.
  - If a network folder is specified, it is necessary to have sufficient privileges in the server.
  - The "application" button can be used to locate the .exe file.
- **Execute:** the program selected in the application dialog is launched when this button is pressed..
- **Break:** aborts program execution.
- **Number of processes:** Sets the number of processes to be launched.
- **Credential Store Account:** Sets the active user of the Credential Store.
- **Check box "more options":** It expands/contracts the options area.
- **Hosts:** Introduce here the list of hosts where you want the processes to run. Host names are separated by blanks. To execute the program in the local machine only, keep the default option "localonly" active or write down its name on this list

Credential Store Tab.

This tab is used to manage user's credential store. If no credential store has been created so far, select "enable create store options" check box to make remaining options available. They are hidden by default since they are only used the first time Deino is initiated.



DeinoMPI 2.0.1

Mpiexec | Credential Store | Cluster | Verify job | Web

Password: login

Location: Select — Registry

Credentials: chema — Remove

Add Credential: Account, Password — Add

Keys: Public Key hash: 7ADDCD65FF7D7B96EA4568FC944 — export

If you believe your keys need to be replaced (compromised, policy, etc) you can do so here. The new public key will need to be distributed to the machines in your cluster.

[ ] enable create -> Create New Keys

[✓] enable create store options

Create a Credential Store — Help — High security — Secure and convenient — Create Credential Store

Password: ○ Password protect private key [● No password]

Encryption: [● Windows ProtectData API] ○ symmetric key ○ no encryption

Location: ○ Removable media [● Registry] ○ Hard drive

Figure A2. Credential Store tab including all options.

In order to create a credential store, the "enable create store options" check box must be selected. Three possibilities arise:

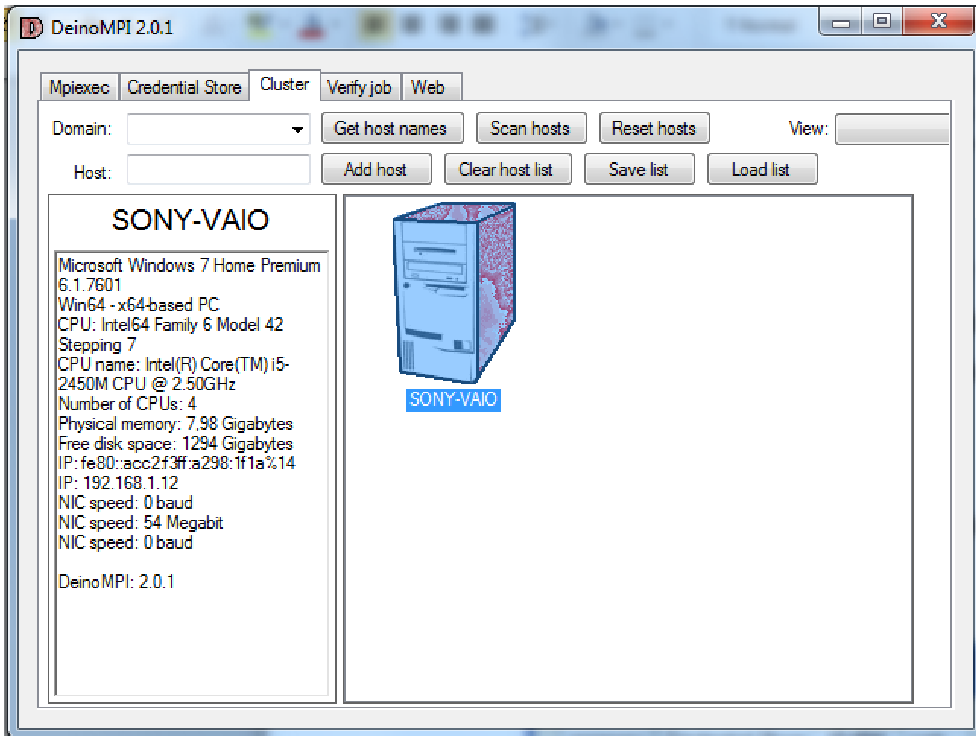
- **"Password":**
  - If this option is selected, the credential store will be protected from access by a password. It is the most secure option but forces the user to introduce the password any time a job has to be launched.
  - If "No password" is selected, the use of MPI is easier but more vulnerable. Without a password any program launched by the user can access the credential store which is not really a problem provided no malicious software is being used.
  - Even with this "No password" option active, the credential store is not available to other users if the encryption option is selected.
- **"Encryption":**

- “Windows ProtectData API” allows encryption of the credential store using the encryption scheme used by Windows for the current user. This ensures the credential store will only be available when the user is validated.
- If a password is selected the “symmetric key” encryption format can be chosen. This encryption is not specific to the user so other user knowing the password could access the store.
- The “no encryption” option is not recommended since it stores the credential store in a plain text file accessible to all users.

- **“Location”:**
  - Take the “Removable media” option to save the store in an external device such as a memory stick. In this case, jobs can only be launched when the device is attached to the computer. This can be the safest option since the user can decide when the credential store is present. Combined with the use of a password and its encryption it can be protected even against loss or robbery.
  - The “Registry” option moves the “Credential Store” to the Windows registry.
  - Finally, it can be stored in the “Hard drive” which turns out to be the most common decision.

Cluster tab

In this tab, the computers in the cluster are displayed and the DeinoMPI version installed in each of them.



```
Microsoft Windows 7 Home Premium
6.1.7601
Win64 - x64-based PC
CPU: Intel64 Family 6 Model 42
Stepping 7
CPU name: Intel(R) Core(TM) i5-
2450M CPU @ 2.50GHz
Number of CPUs: 4
Physical memory: 7,98 Gigabytes
Free disk space: 1294 Gigabytes
IP: fe80::acc2:f3ff:a298:1f1a%14
IP: 192.168.1.12
NIC speed: 0 baud
NIC speed: 54 Megabit
NIC speed: 0 baud

DeinoMPI: 2.0.1
```

Figure A3. Cluster tab – Big icons view.

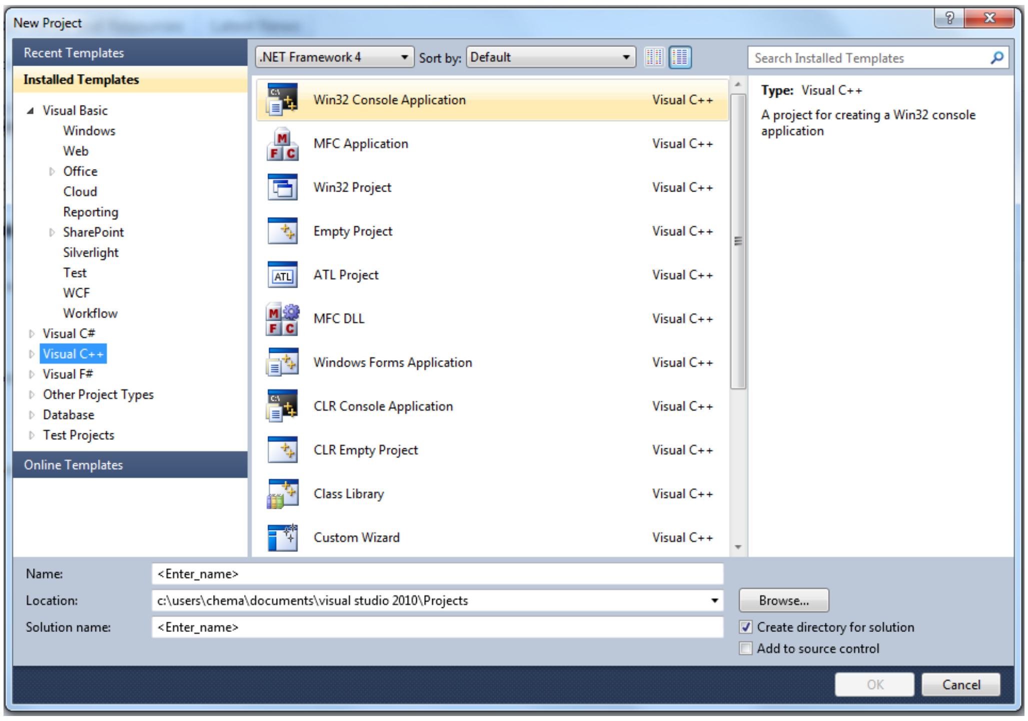
More hosts can be added writing down their name of can be found automatically within the selected domain.

Deino MPI manual. Available at: http://mpi.deino.net/manual.htm

Appendix B: Project Configuration in Visual Studio 2010

In this section we will describe the same configuration process but for the 2010 version of Microsoft Visual Studio. Configuration in more recent versions of Visual Studio is analogous.

- Generate a **new project and solution**. They may have both the same name:



Type: Visual C++

A project for creating a Win32 console application

Name: <Enter_name>

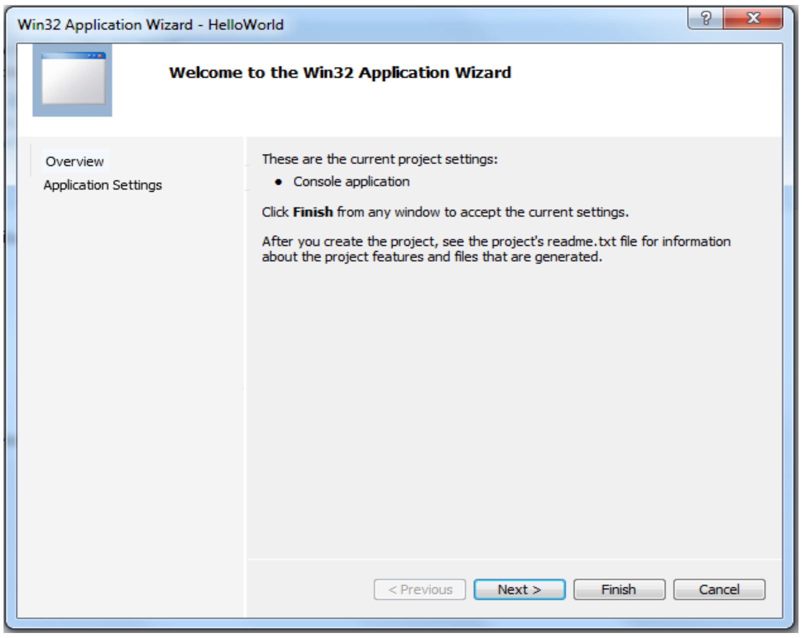
Location: c:\users\chema\documents\visual studio 2010\Projects

Solution name: <Enter_name>

☑ Create directory for solution

☐ Add to source control

- Set it as **empty project**:



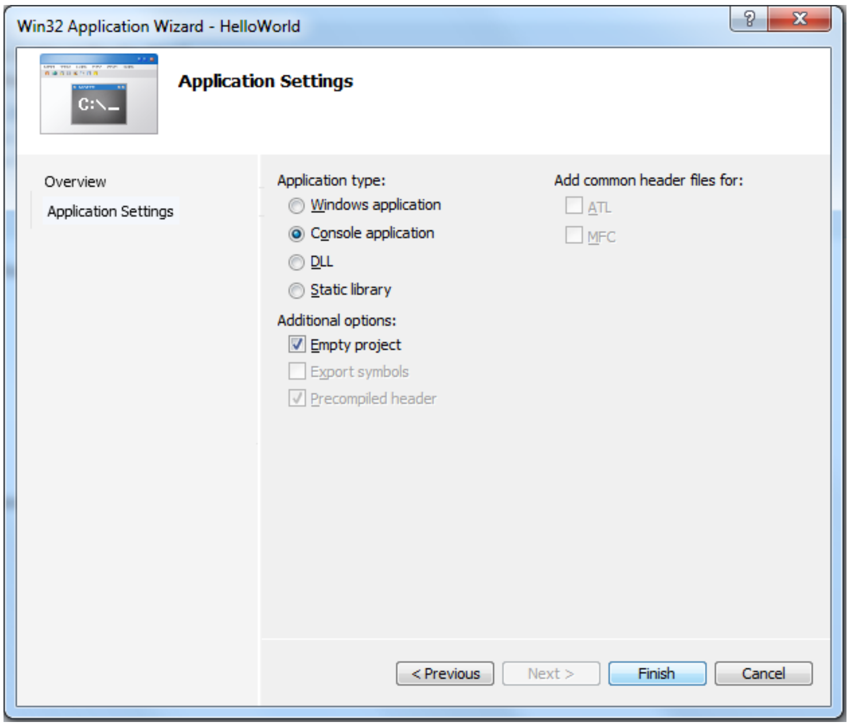
Welcome to the Win32 Application Wizard

These are the current project settings:

- Console application

Click **Finish** from any window to accept the current settings.

After you create the project, see the project's readme.txt file for information about the project features and files that are generated.



Win32 Application Wizard - HelloWorld

Application Settings

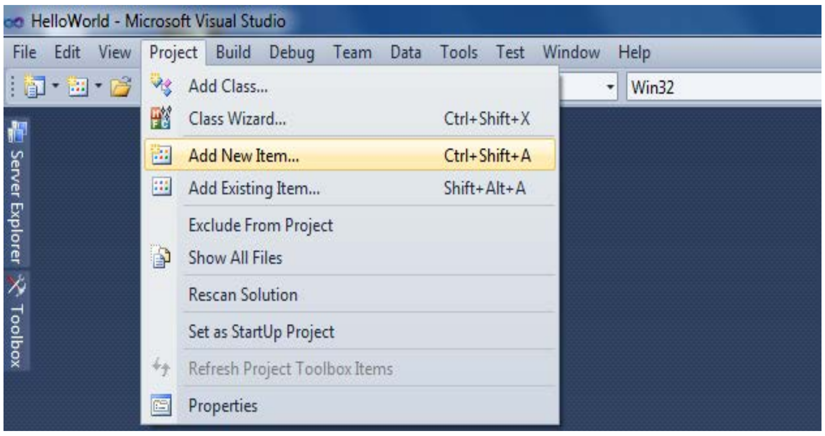
Overview
Application Settings

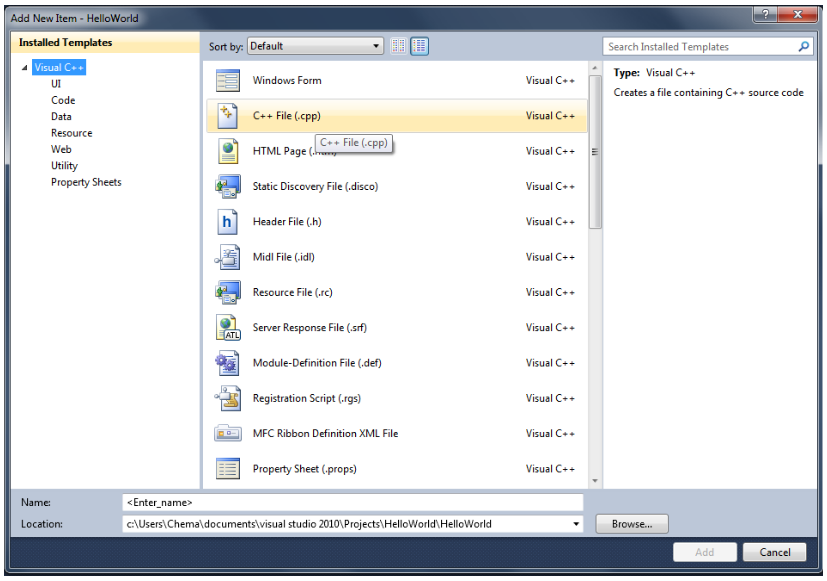
Application type:
- Windows application
- [✓ Console application]
- DLL
- Static library

Additional options:
- [✓ Empty project]
- Export symbols
- [✓ Precompiled header]

Add common header files for:
- ATL
- MFC

- Once created both the Project and solution, add a code file as **new item**:





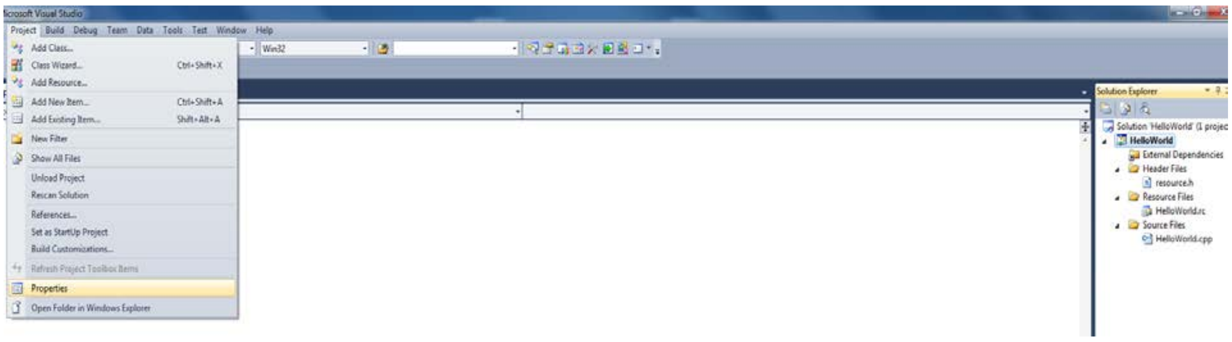
Add New Item - HelloWorld

Windows Form	Visual C++
C++ File (.cpp)	Visual C++
HTML Page (.htm)	Visual C++
Static Discovery File (.disco)	Visual C++
Header File (.h)	Visual C++
Midl File (.idl)	Visual C++
Resource File (.rc)	Visual C++
Server Response File (.srf)	Visual C++
Module-Definition File (.def)	Visual C++
Registration Script (.rgs)	Visual C++
MFC Ribbon Definition XML File	Visual C++
Property Sheet (.props)	Visual C++

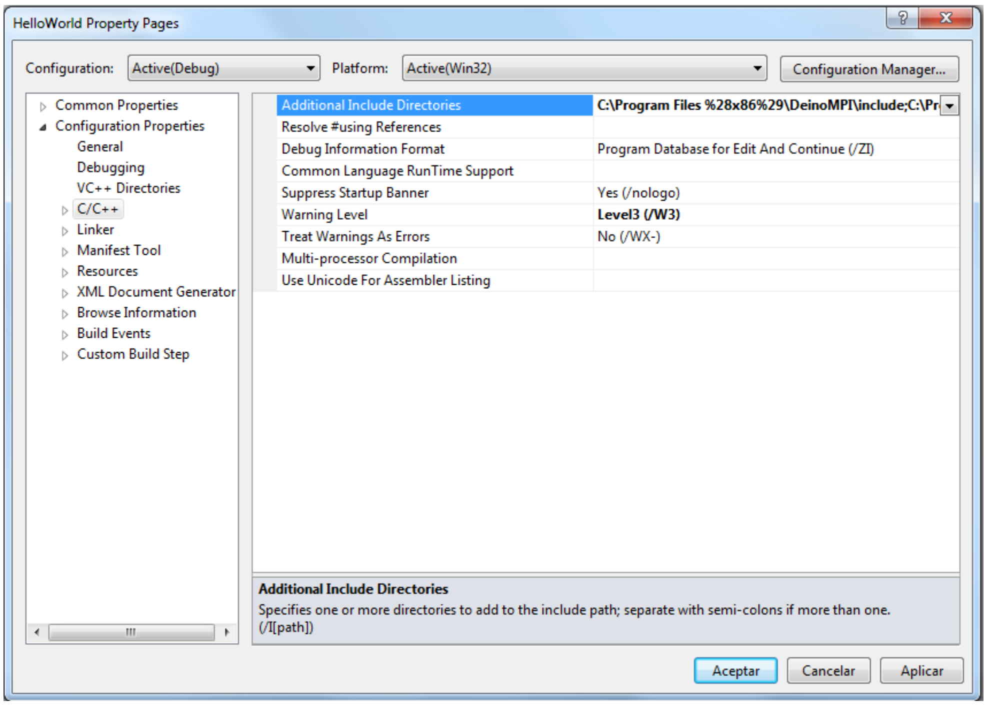
Type: Visual C++
Creates a file containing C++ source code

Name: <Enter_name>
Location: c:\Users\Chema\documents\visual studio 2010\Projects\HelloWorld\HelloWorld

- Now, and never before, the Project settings are entered ("*Properties*"):



1. In the **C/C++** section we must enter the route to the folder where the header MPI files are located ("*Additional Include Directories*"). By default the *\Archivos de Programa (x86)\DeinoMPI\include* is assumed:



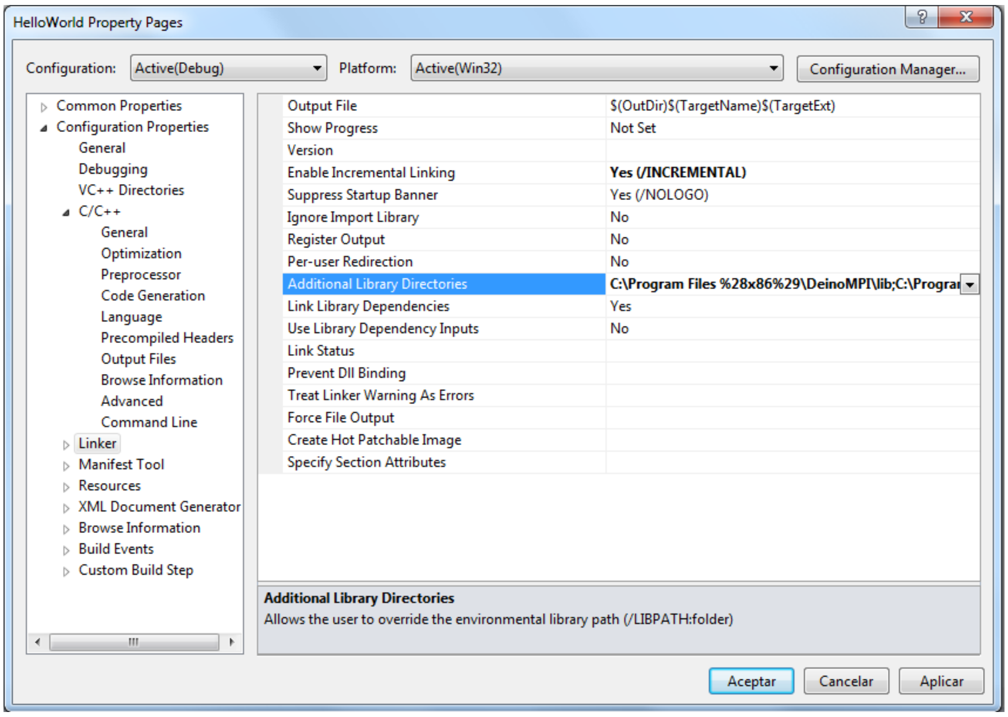
HelloWorld Property Pages

Configuration: Active(Debug) Platform: Active(Win32) Configuration Manager...

Additional Include Directories	C:\Program Files %28x86%29\DeinoMPI\include;C:\Pr
Resolve #using References	
Debug Information Format	Program Database for Edit And Continue (/ZI)
Common Language RunTime Support	
Suppress Startup Banner	Yes (/nologo)
Warning Level	**Level3 (/W3)**
Treat Warnings As Errors	No (/WX-)
Multi-processor Compilation	
Use Unicode For Assembler Listing	

Additional Include Directories
Specifies one or more directories to add to the include path; separate with semi-colons if more than one. (/I[path])

2. In the **Linker** section we must enter the route to the folder where the MPI libraries are located ("*Additional Library Directories*"). By default *\Archivos de Programa (x86)\DeinoMPI\lib* is assumed:



HelloWorld Property Pages

Configuration: Active(Debug) Platform: Active(Win32) Configuration Manager...

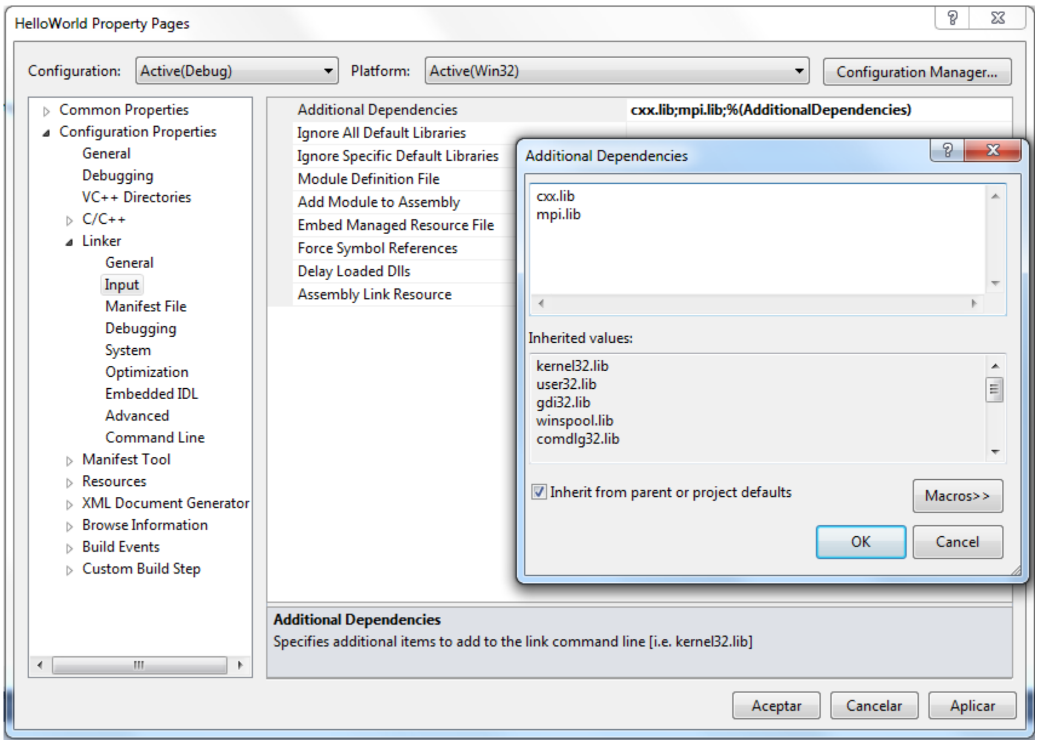
- Common Properties
- Configuration Properties
  - General
  - Debugging
  - VC++ Directories
  - C/C++
    - General
    - Optimization
    - Preprocessor
    - Code Generation
    - Language
    - Precompiled Headers
    - Output Files
    - Browse Information
    - Advanced
    - Command Line
  - Linker
  - Manifest Tool
  - Resources
  - XML Document Generator
  - Browse Information
  - Build Events
  - Custom Build Step

Output File	$(OutDir)$(TargetName)$(TargetExt)
Show Progress	Not Set
Version	
Enable Incremental Linking	**Yes (/INCREMENTAL)**
Suppress Startup Banner	Yes (/NOLOGO)
Ignore Import Library	No
Register Output	No
Per-user Redirection	No
Additional Library Directories	**C:\Program Files %28x86%29\DeinoMPI\lib;C:\Prograı**
Link Library Dependencies	Yes
Use Library Dependency Inputs	No
Link Status	
Prevent Dll Binding	
Treat Linker Warning As Errors	
Force File Output	
Create Hot Patchable Image	
Specify Section Attributes	

Additional Library Directories
Allows the user to override the environmental library path (/LIBPATH:folder)

Aceptar Cancelar Aplicar

3. In the **Linker** section, in the input entry ("*Input*") the "**cxx.lib**" y "**mpi.lib**" files must be added as additional dependencies:



HelloWorld Property Pages

Configuration: Active(Debug) Platform: Active(Win32) Configuration Manager...

- Common Properties
- Configuration Properties
  - General
  - Debugging
  - VC++ Directories
  - C/C++
  - Linker
    - General
    - Input
    - Manifest File
    - Debugging
    - System
    - Optimization
    - Embedded IDL
    - Advanced
    - Command Line
  - Manifest Tool
  - Resources
  - XML Document Generator
  - Browse Information
  - Build Events
  - Custom Build Step

Additional Dependencies	**cxx.lib;mpi.lib;%(AdditionalDependencies)**
Ignore All Default Libraries	
Ignore Specific Default Libraries	
Module Definition File	
Add Module to Assembly	
Embed Managed Resource File	
Force Symbol References	
Delay Loaded Dlls	
Assembly Link Resource	

Additional Dependencies

```
cxx.lib
mpi.lib
```

Inherited values:

```
kernel32.lib
user32.lib
gdi32.lib
winspool.lib
comdlg32.lib
```

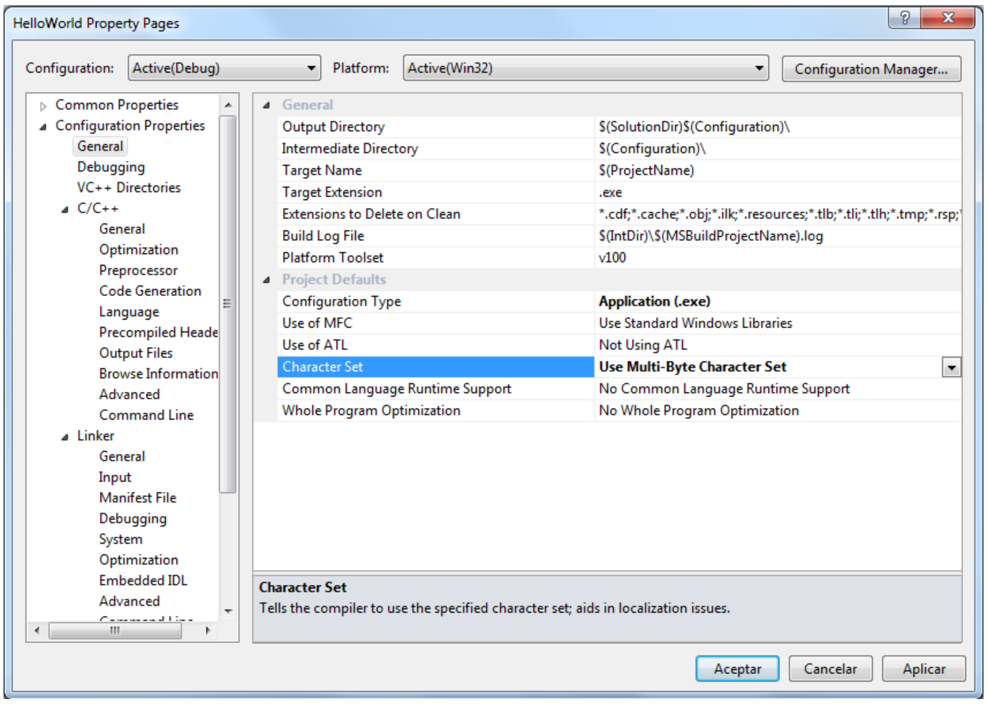
☑ Inherit from parent or project defaults

Macros>> OK Cancel

Additional Dependencies
Specifies additional items to add to the link command line [i.e. kernel32.lib]

Aceptar Cancelar Aplicar

4. In the **General** section the **Multi-Byte** set of characters ("*Characrer Set*") must be selected:



General

Output Directory	$(SolutionDir)$(Configuration)\
Intermediate Directory	$(Configuration)\
Target Name	$(ProjectName)
Target Extension	.exe
Extensions to Delete on Clean	*.cdf;*.cache;*.obj;*.ilk;*.resources;*.tlb;*.tli;*.tlh;*.tmp;*.rsp;
Build Log File	$(IntDir)\$(MSBuildProjectName).log
Platform Toolset	v100

Project Defaults

Configuration Type	**Application (.exe)**
Use of MFC	Use Standard Windows Libraries
Use of ATL	Not Using ATL
Character Set	**Use Multi-Byte Character Set**
Common Language Runtime Support	No Common Language Runtime Support
Whole Program Optimization	No Whole Program Optimization

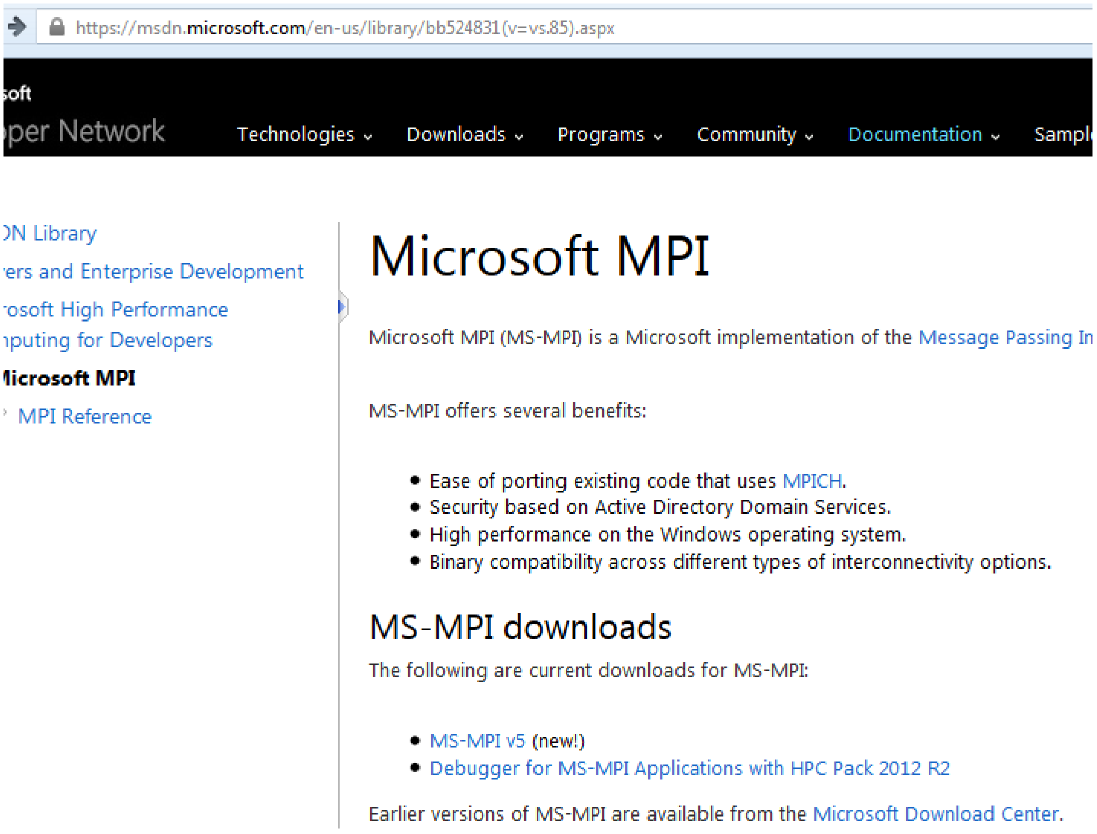
Character Set
Tells the compiler to use the specified character set; aids in localization issues.

- Finally enter code in the selected source file and build the project.

Appendix C: Configuration of MS-MPI.

DeinoMPI is hard to configure in some systems and it may not eventually work. As an alternative we can install and configure the Microsoft distribution of MPI. It doesn't provide a graphical interface but from the command line everything can be done. Take the following steps to get it to work:

- Download MS-MPI v5 from its web location:



Microsoft MPI

Microsoft MPI (MS-MPI) is a Microsoft implementation of the Message Passing In

MS-MPI offers several benefits:

- Ease of porting existing code that uses MPICH.
- Security based on Active Directory Domain Services.
- High performance on the Windows operating system.
- Binary compatibility across different types of interconnectivity options.

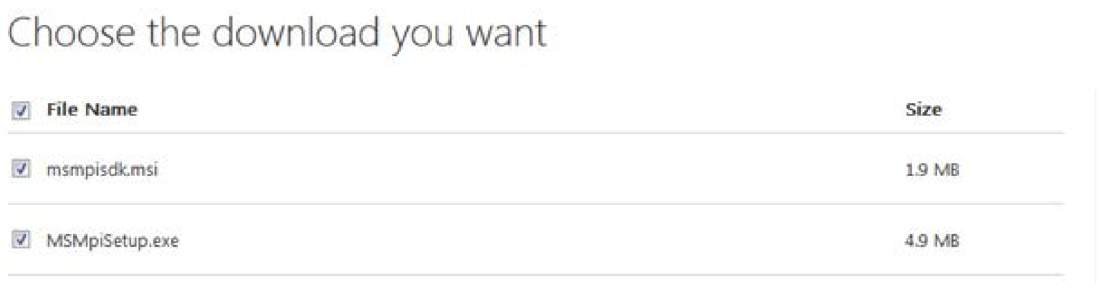
MS-MPI downloads

The following are current downloads for MS-MPI:

- MS-MPI v5 (new!)
- Debugger for MS-MPI Applications with HPC Pack 2012 R2

Earlier versions of MS-MPI are available from the Microsoft Download Center.

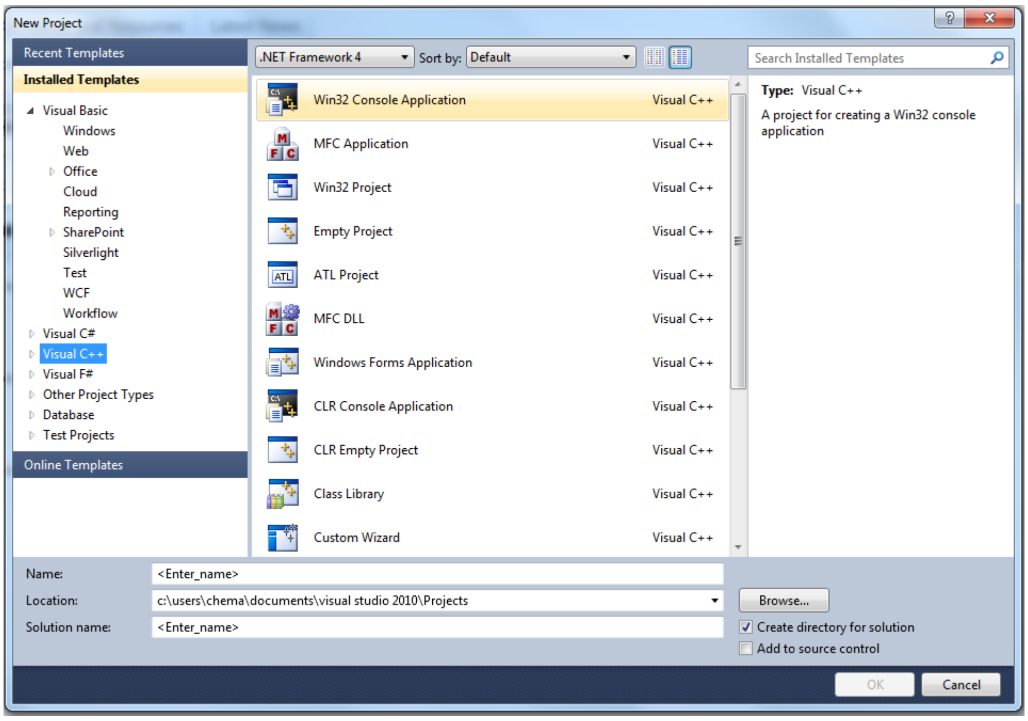
- There are two files and both have to be downloaded and installed:


Choose the download you want

☑ File Name	Size
☑ msmpisdk.msi	1.9 MB
☑ MSMpiSetup.exe	4.9 MB

- Each package creates a new folder: Program Files > Microsoft MPI and Program Files > Microsoft SDKs > MPI.

- Generate a **new MS Visual Studio project and solution**. They may have both the same name:


New Project

Recent Templates

Installed Templates

- Visual Basic
  - Windows
  - Web
  - Office
  - Cloud
  - Reporting
  - SharePoint
  - Silverlight
  - Test
  - WCF
  - Workflow
- Visual C#
- Visual C++
- Visual F#
- Other Project Types
- Database
- Test Projects

Online Templates

.NET Framework 4 Sort by: Default

Search Installed Templates

Win32 Console Application	Visual C++
MFC Application	Visual C++
Win32 Project	Visual C++
Empty Project	Visual C++
ATL Project	Visual C++
MFC DLL	Visual C++
Windows Forms Application	Visual C++
CLR Console Application	Visual C++
CLR Empty Project	Visual C++
Class Library	Visual C++
Custom Wizard	Visual C++

Type: Visual C++

A project for creating a Win32 console application

Name: <Enter_name>

Location: c:\users\chema\documents\visual studio 2010\Projects

Solution name: <Enter_name>

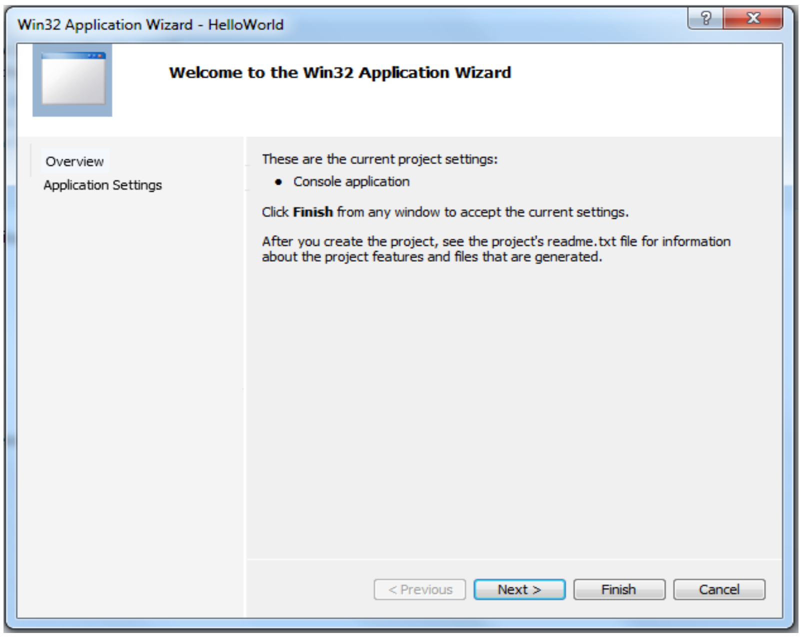
Browse...

☑ Create directory for solution

☐ Add to source control

OK Cancel

- Set it as **empty project**:


Win32 Application Wizard - HelloWorld

Welcome to the Win32 Application Wizard

Overview

Application Settings

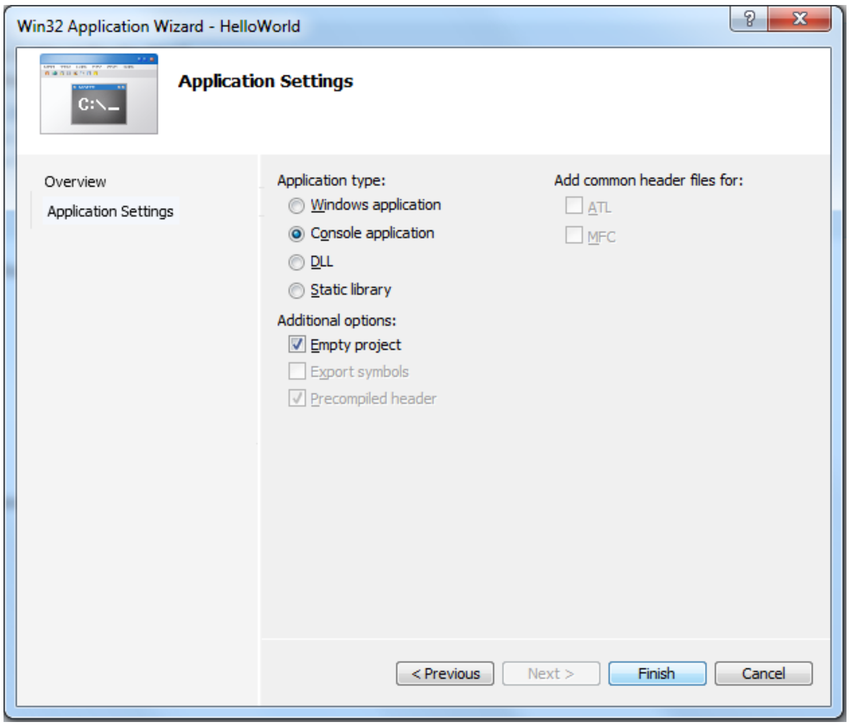
These are the current project settings:

- Console application

Click **Finish** from any window to accept the current settings.

After you create the project, see the project's readme.txt file for information about the project features and files that are generated.

< Previous Next > Finish Cancel


Win32 Application Wizard - HelloWorld

Application Settings

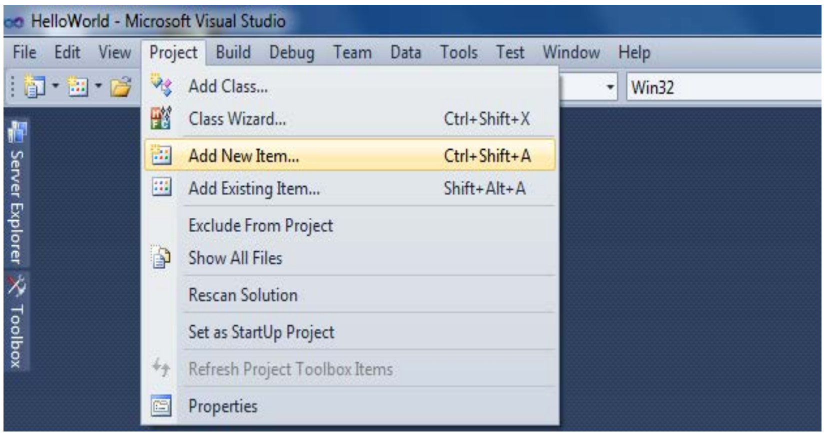
Overview
Application Settings

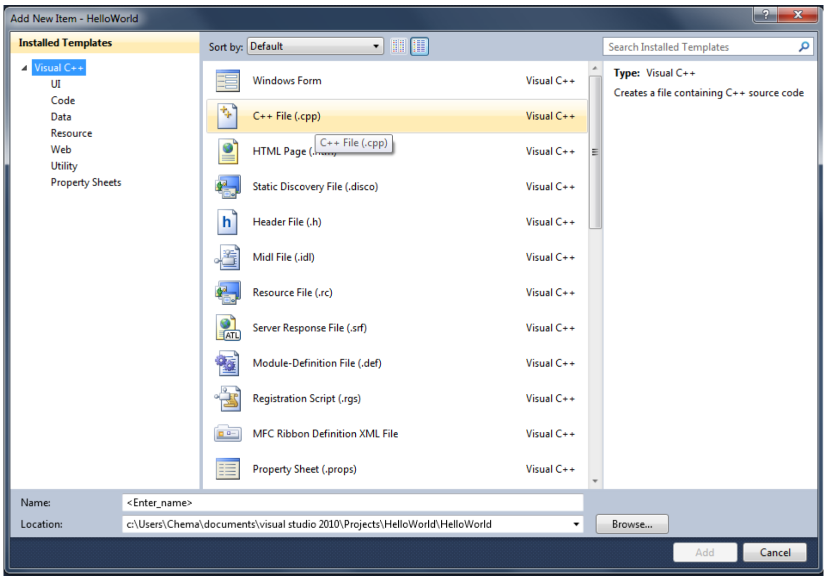
Application type:
Windows application
[✓ Console application]
DLL
Static library

Additional options:
[✓ Empty project]
Export symbols
Precompiled header

Add common header files for:
ATL
MFC

- Once created both the Project and solution, add a code file as **new item**:


HelloWorld - Microsoft Visual Studio
File Edit View Project Build Debug Team Data Tools Test Window Help
Win32
Server Explorer
Toolbox
Add Class...
Class Wizard... Ctrl+Shift+X
Add New Item... Ctrl+Shift+A
Add Existing Item... Shift+Alt+A
Exclude From Project
Show All Files
Rescan Solution
Set as StartUp Project
Refresh Project Toolbox Items
Properties


Add New Item - HelloWorld

Installed Templates
- Visual C++
  - UI
  - Code
  - Data
  - Resource
  - Web
  - Utility
  - Property Sheets

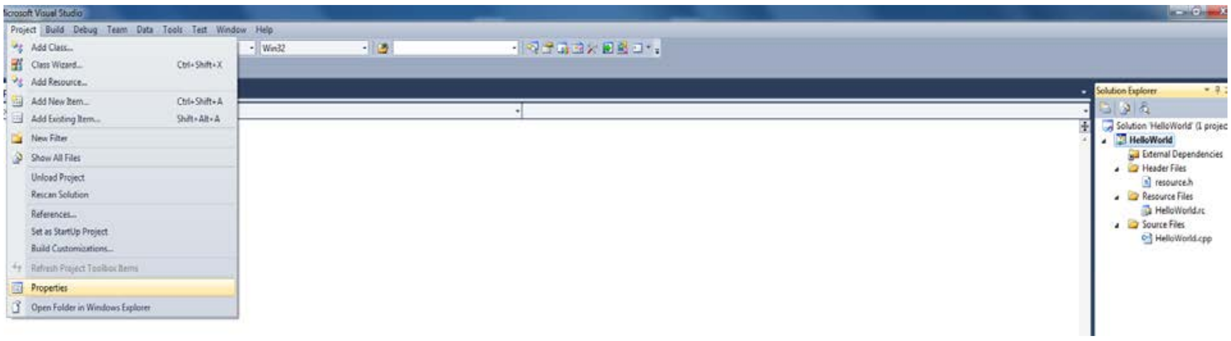
Sort by: Default

Windows Form	Visual C++
C++ File (.cpp)	Visual C++
HTML Page (.htm)	Visual C++
Static Discovery File (.disco)	Visual C++
Header File (.h)	Visual C++
Midl File (.idl)	Visual C++
Resource File (.rc)	Visual C++
Server Response File (.srf)	Visual C++
Module-Definition File (.def)	Visual C++
Registration Script (.rgs)	Visual C++
MFC Ribbon Definition XML File	Visual C++
Property Sheet (.props)	Visual C++

Type: Visual C++
Creates a file containing C++ source code

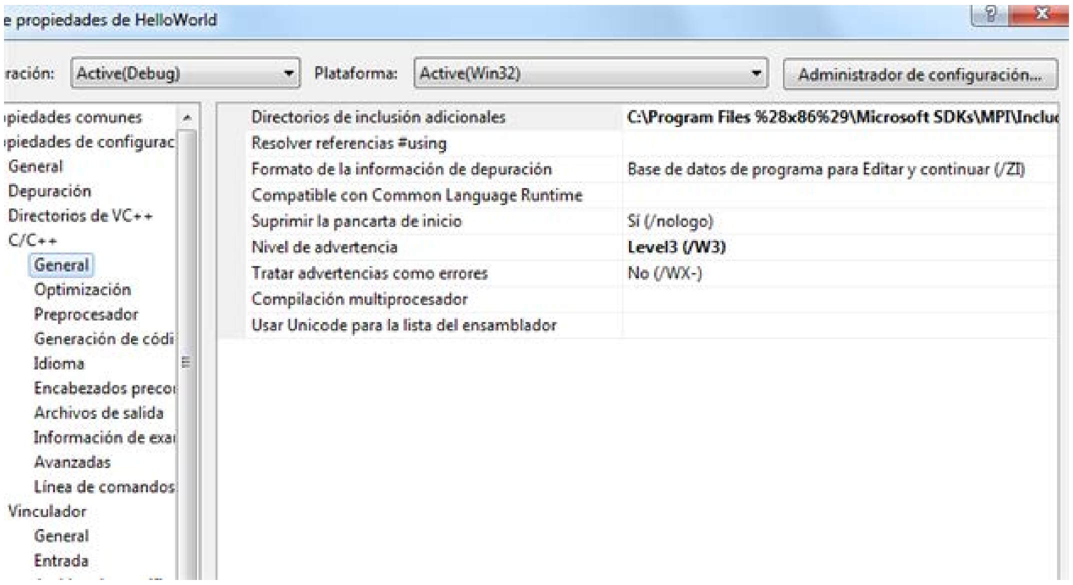
Name: <Enter_name>
Location: c:\Users\Chema\documents\visual studio 2010\Projects\HelloWorld\HelloWorld

- Now, and never before, the Project settings are entered ("*Properties*"):



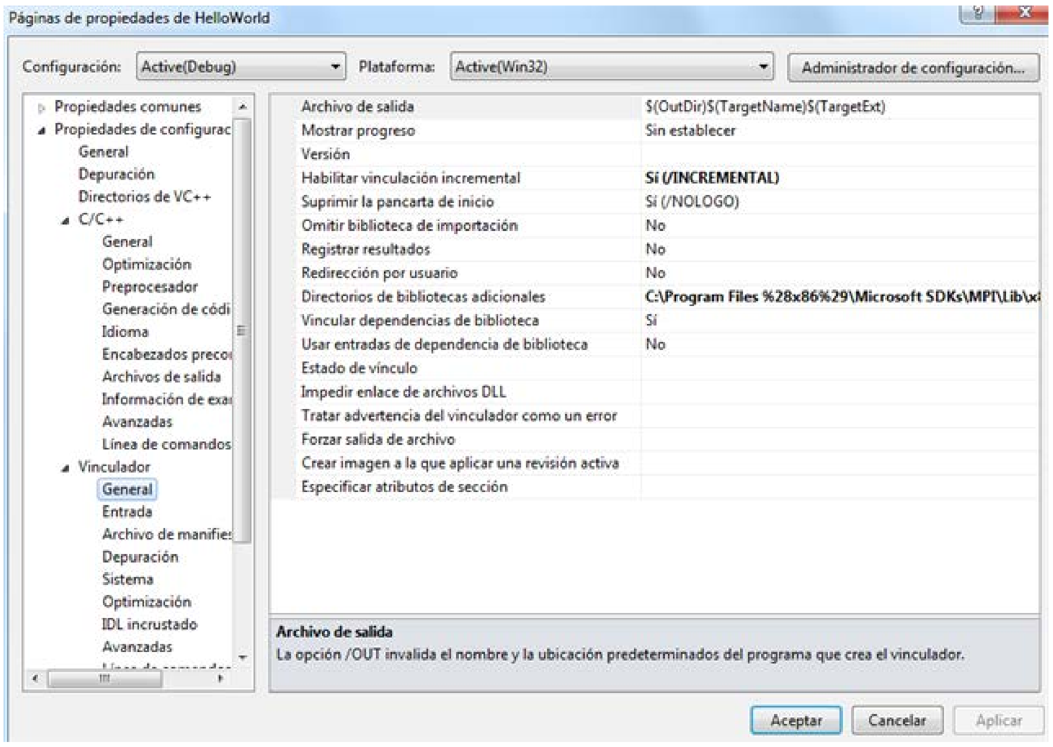
- When reaching the project configuration options proceed as follows:

1. Set the new additional include folder.



Directorios de inclusión adicionales	C:\Program Files %28x86%29\Microsoft SDKs\MPI\Incluc
Resolver referencias #using	
Formato de la información de depuración	Base de datos de programa para Editar y continuar (/ZI)
Compatible con Common Language Runtime	
Suprimir la pancarta de inicio	Sí (/nologo)
Nivel de advertencia	Level3 (/W3)
Tratar advertencias como errores	No (/WX-)
Compilación multiprocesador	
Usar Unicode para la lista del ensamblador	

2. Similarly set the new lib folder. Under lib choose the folder that matches your development (x86 for 32 bit applications or x64 for 64 bit ones).

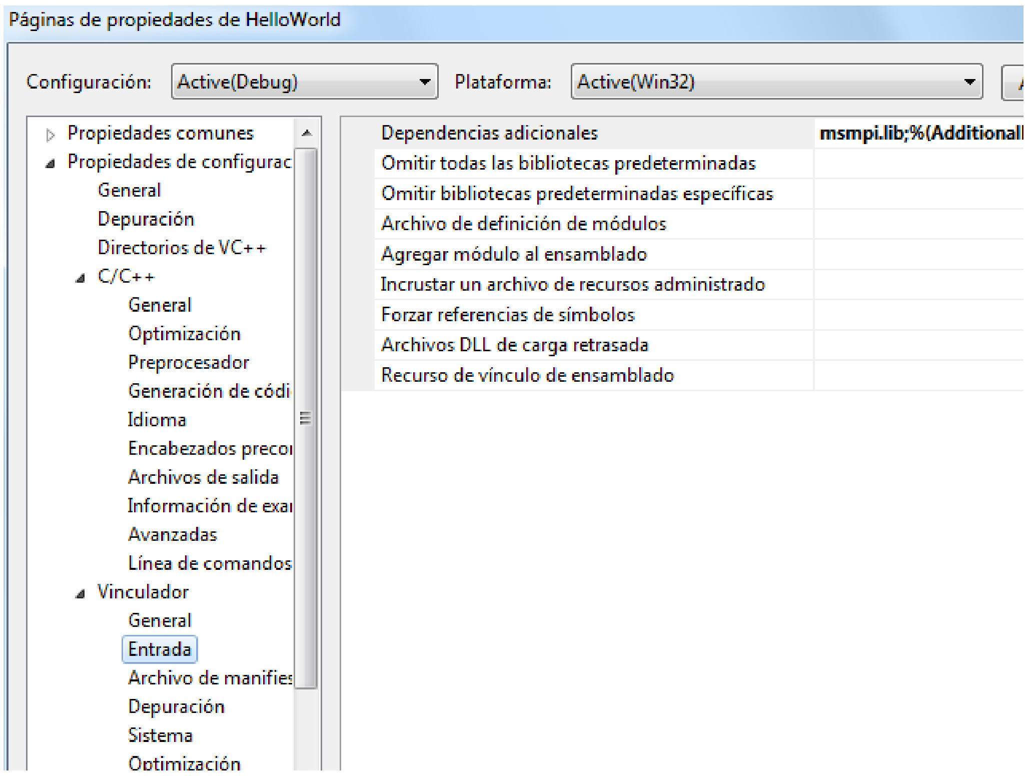


Configuración: Active(Debug) Plataforma: Active(Win32)

Archivo de salida	$(OutDir)$(TargetName)$(TargetExt)
Mostrar progreso	Sin establecer
Versión	
Habilitar vinculación incremental	**Sí (/INCREMENTAL)**
Suprimir la pancarta de inicio	Sí (/NOLOGO)
Omitir biblioteca de importación	No
Registrar resultados	No
Redirección por usuario	No
Directorios de bibliotecas adicionales	**C:\Program Files %28x86%29\Microsoft SDKs\MPI\Lib\x**
Vincular dependencias de biblioteca	Sí
Usar entradas de dependencia de biblioteca	No
Estado de vínculo	
Impedir enlace de archivos DLL	
Tratar advertencia del vinculador como un error	
Forzar salida de archivo	
Crear imagen a la que aplicar una revisión activa	
Especificar atributos de sección	

Archivo de salida
La opción /OUT invalida el nombre y la ubicación predeterminados del programa que crea el vinculador.

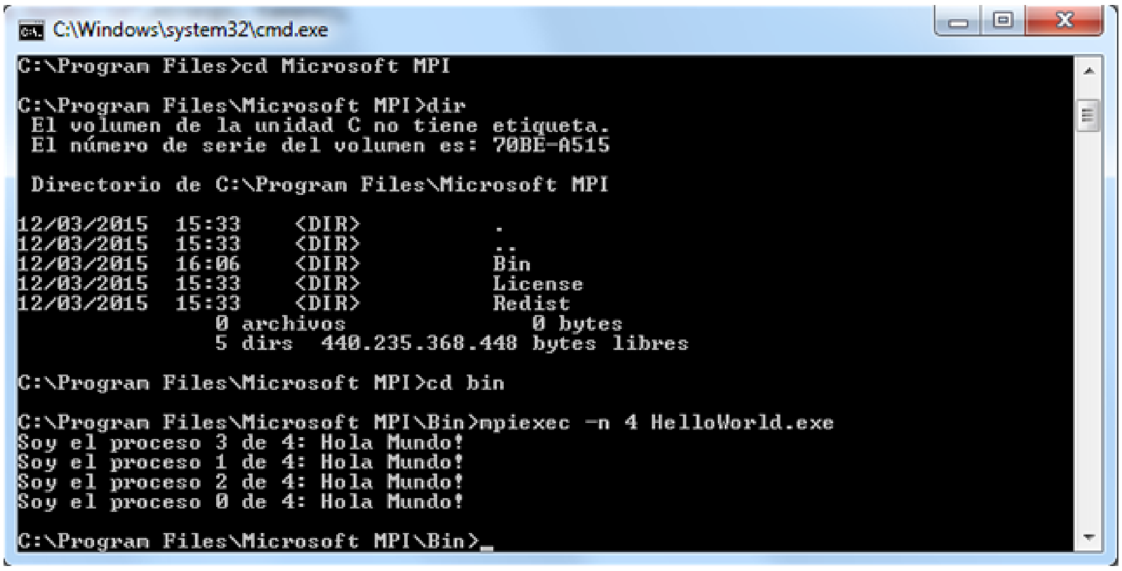
3. Set also the new library file.



Configuración: Active(Debug) Plataforma: Active(Win32)

Dependencias adicionales	**msmpi.lib;%(Additional**
Omitir todas las bibliotecas predeterminadas	
Omitir bibliotecas predeterminadas específicas	
Archivo de definición de módulos	
Agregar módulo al ensamblado	
Incrustar un archivo de recursos administrado	
Forzar referencias de símbolos	
Archivos DLL de carga retrasada	
Recurso de vínculo de ensamblado	

4. When all these parts have been configured the solution can be built as usual. In order to execute the program, the .exe file and MPI's launcher must be in the same folder or either the path configured accordingly. The launcher is mpiexec.exe and is placed in Program Files > Microsoft MPI > bin. Write down mpiexec –n np program.exe, where np is the number of processes to be launched.



```
C:\Program Files>cd Microsoft MPI

C:\Program Files\Microsoft MPI>dir
 El volumen de la unidad C no tiene etiqueta.
 El número de serie del volumen es: 70BE-A515

 Directorio de C:\Program Files\Microsoft MPI

12/03/2015  15:33    <DIR>          .
12/03/2015  15:33    <DIR>          ..
12/03/2015  16:06    <DIR>          Bin
12/03/2015  15:33    <DIR>          License
12/03/2015  15:33    <DIR>          Redist
               0 archivos              0 bytes
               5 dirs  440.235.368.448 bytes libres

C:\Program Files\Microsoft MPI>cd bin

C:\Program Files\Microsoft MPI\Bin>mpiexec -n 4 HelloWorld.exe
Soy el proceso 3 de 4: Hola Mundo!
Soy el proceso 1 de 4: Hola Mundo!
Soy el proceso 2 de 4: Hola Mundo!
Soy el proceso 0 de 4: Hola Mundo!

C:\Program Files\Microsoft MPI\Bin>_
```