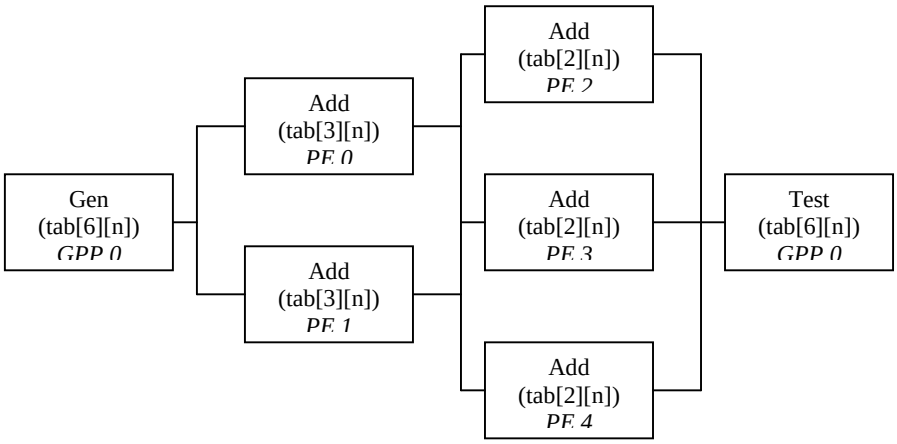
SMECY: IR proposition

Author: Rémi Barrère
11/05/2011

Application description

This application, very simple, is used to describe the different pragmas that are needed to map an application on a parallel architecture.

This application begins with the first « task » that will generate an 2D array of 6 x n values, executed on the GPP0. This array is then sent to 2 differents processors called PE0 and PE1, that will call the *Add* function to all values they received (one half each). Then these processors send their results to 3 different processors called PE2, PE3 and PE4, that will do the same job (one third each). Once finished, they all send their results to the GPP0 that will check all values are OK.



Pragmas used

The different pragmas that can be used in the SMECY project :

- OpenMP pragmas for parallelism expression (or a subset),
- *BlueBee* map pragma to express on which processor the following function will be executed,
- A new pragma to express data transfer between different processors.

This new pragma can be compared to the Par4All API copy primitives, like P4A_copy_from_accel_3d (for a 3D array). The pragma looks like:

```
#pragma communication  (     element_size,
                             d1_size,
                             d1_block size,
                             d1_dest_offset,
                             d1_org_offset,
                             …
                             [dest processor][ dest memory]
                             [org processor][org memory]
                       ).
```

Parameters :

- *element_size*: is the size of one element of the array in byte,
-
- *d1_size* : is the number of elements in the array,
- *d1_block size* : is the number of element to transfer,

Repeated for each array dimension

- *d1_dest_offset* : is element order to start the transfer from on the destination memory,
- *d1_org_offset* : is element order to start the transfer from on the origin memory,
-
- *dest processor* : is the name of the processor owning the destination memory
- *dest memory :* is the name of the destination memory,
- *org processor* : is the name of the processor owning the origin memory,
- *org memory :* is the name of the origin memory.

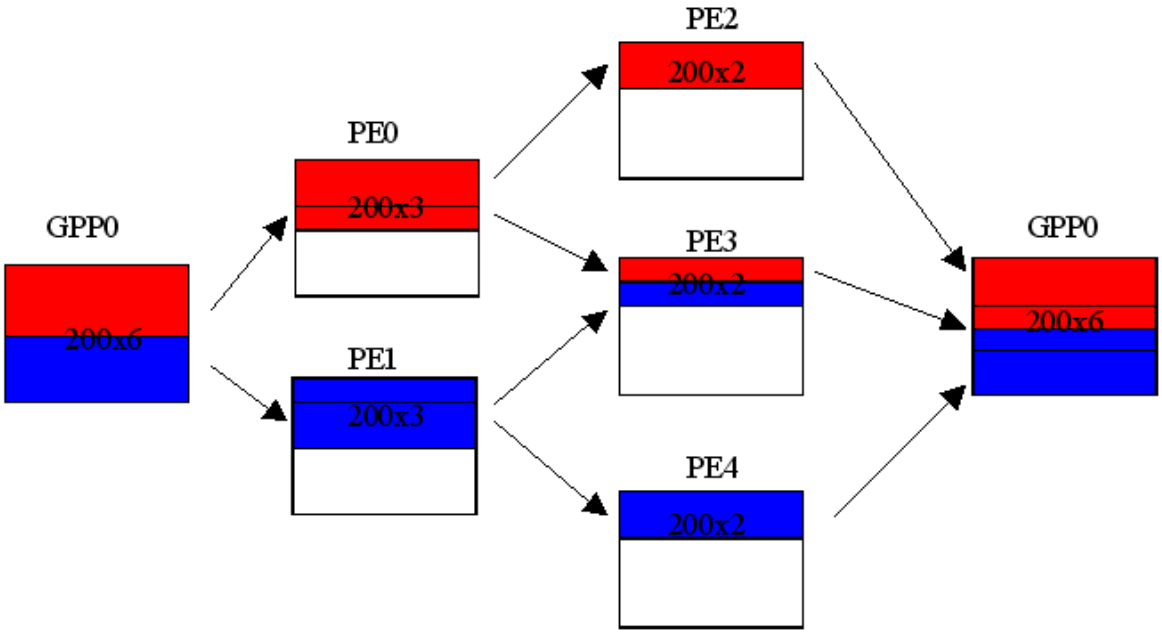
The [processor] parameter can be replaced by a [*] in case of the use of a shared memory.

Code sample

The example of code is given as an example that can be used as a draft document that will be iteratively be improved. The goal is to converge towards a IR that will use the maximum of already existing pragmas, and add only missing information.

The communications are described more in detail here :

- GPP0 => PE0 : transfer of 200x3 bytes with no origin offset and no destination offset,
- GPP0 => PE1 : transfer of 200x3 bytes with origin offset of 200x3 and no destination offset,
- PE0 => PE2 : transfer of 200x2 bytes with no origin offset and no destination offset,
- PE0 => PE3 : transfer of 200x1 bytes with origin offset of 200x2 and no destination offset,
- PE1 => PE3 : transfer of 200x1 bytes with no origin offset and destination offset of 200x1,
- PE1 => PE4 : transfer of 200x2 bytes with origin offset of 200x1 and no destination offset,
- PE2 => GPP0 : transfer of 200x2 bytes with no origin offset and no destination offset,
- PE3 => GPP0 : transfer of 200x2 bytes with no origin offset and destination offset of 200x2,
- PE4 => GPP0 : transfer of 200x2 bytes with no origin offset and destination offset of 200x4.



```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/* function Gen */
void Gen(int *out, int size)
{
	int i=0;

	// Can be parallel
	#pragma omp parallel for
	for (i = 0; i < size; i++)
	{
		out [i] = 0;
	}
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/* function Add */
void Add(int* in, int *out, int size)
{
	int i=0;

	// Can be parallel
	#pragma omp parallel for
	for (i = 0; i < size; i++)
	{
		out [i] = in [i] + 1;
	}
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/* function Test */
void Test (int* in, int size)
{
	int i=0;

	// Can be parallel
	#pragma omp parallel for
	for (i = 0; i < size; i++)
	{
		if (in [i] != 2)
			exit(-1) ;
	}
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/* main */
int main(int argc, char* argv[])
{
	int tab[6][200];

	// Gen is mapped on GPP 0
	#pragma map call_hw GPP 0 arg(0,200*6)
	Gen(tab, 200*6);

	// Communication from GPP 0 to PE 0
	#pragma communication (1, 3, 3, 0, 0, 200, 200, 0, 0, [PE 0][local], [GPP 0])
	// Communication from GPP 0 to PE 1
	#pragma communication (1, 3, 3, 0, 3, 200, 200, 0, 0, [PE 1][local], [GPP 0])

	// Parallelism on different processors
	#pragma omp parallel
	{
		{
			#pragma omp section
			#pragma map call_hw PE 0 arg(0,200*3) arg(1,200*3)
			Add(tab, tab, 200*3);
		}
		{
			#pragma omp section
```

```
                        #pragma map call_hw PE 1 arg(0,200*3) arg(1,200*3)
                        Add(tab, tab, 200*3);
                }
        }

        // Communication from PE 0 to PE 2
        #pragma communication (1, 2, 2, 0, 0, 200, 200, 0, 0, [PE 2][local], [PE 0][local])
        // Communication from PE 0 to PE 3
        #pragma communication (1, 1, 1, 0, 2, 200, 200, 0, 0, [PE 3][local], [PE 0][local])
        // Communication from PE 1 to PE 3
        #pragma communication (1, 1, 1, 1, 0, 200, 200, 0, 0, [PE 3][local], [PE 1][local])
        // Communication from PE 1 to PE 4
        #pragma communication (1, 2, 2, 0, 1, 200, 200, 0, 0, [PE 4][local], [PE 1][local])

        // Parallelism on different processors
        #pragma omp parallel
        {
                {
                        #pragma omp section
                        #pragma map call_hw PE 2 arg(0,200*2) arg(1,200*2)
                        Add(tab, tab, 200*2);
                }
                {
                        #pragma omp section
                        #pragma map call_hw PE 3 arg(0,200*2) arg(1,200*2)
                        Add(tab, tab, 200*2);
                }
                {
                        #pragma omp section
                        #pragma map call_hw PE 4 arg(0,200*2) arg(1,200*2)
                        Add(tab, tab, 200*2);
                }
        }

        // Communication from PE 2 to GPP 0
        #pragma communication (1, 200, 200, 0, 0, 2, 2, 0, 0, [GPP 0], [PE 2][local])
        // Communication from PE 3 to GPP 0
        #pragma communication (1, 200, 200, 0, 0, 2, 2, 2, 0, [GPP 0], [PE 3][local])
        // Communication from PE 4 to GPP 0
        #pragma communication (1, 200, 200, 0, 0, 2, 2, 4, 0, [GPP 0], [PE 4][local])


        // Test is mapped on GPP 0
        #pragma map call_hw GPP 0 arg(0,200*6)
        Test(tab, 200*6);

        return 0;
}
```