Asian Option Pricing on cluster of GPUs: First Results

(ANR project « GCPMF »)

S. Vialle – SUPELEC
L. Abbas-Turki – ENPC

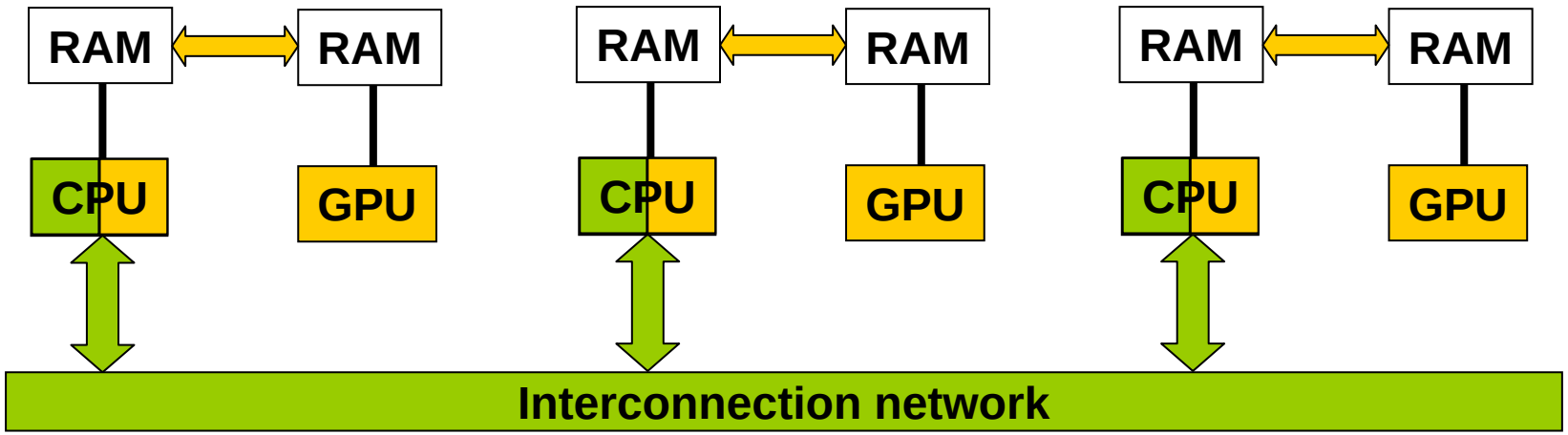
With the help of P. Mercier (SUPELEC).
Previous work of G. Noaje March-June 2008.

1 – Building a GPU cluster for experimentations…

1.1 – Objectives & Strategy

1. Build a 16-node cluster to experiment distributed computing on GPU
2. With a multi-core CPU and a GPU on each node
3. Choose hardware to support asynchronous communications and maximal overlapping strategy:

Parallelization and overlapping of: GPU computation, CPU-GPU communications, CPU computations, and CPU-CPU communications (across the interconnection network).

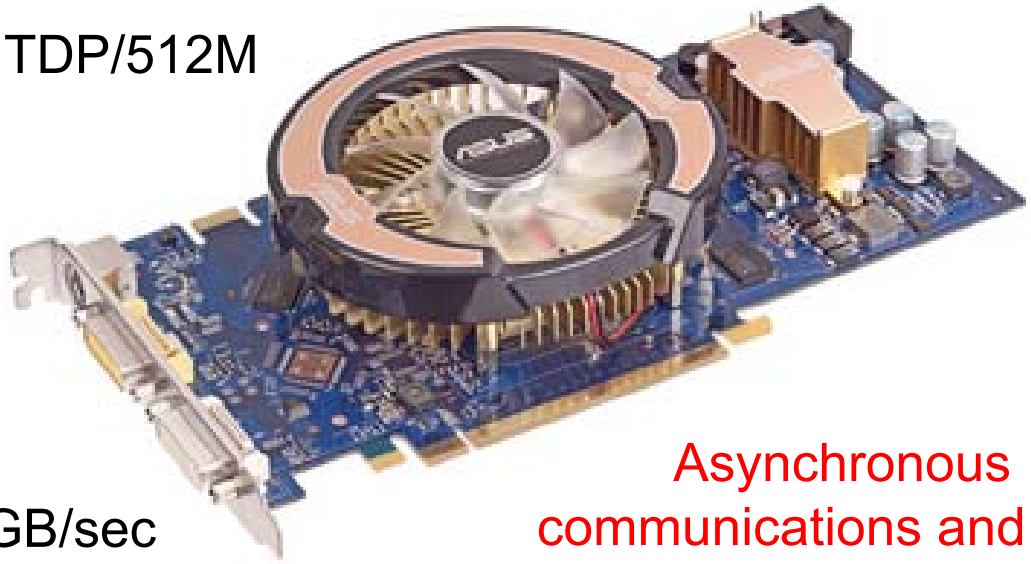


1.2 - Hardware choice

GPU on each node: ASUS GeForce 8800 GT

Product model: EN8800GT/G/HTDP/512M

- Multiprocessors: 14
- Stream processors: 112
- Core clock: 600 MHz
- Memory clock: 900 MHz
- Memory amount: 512 MB
- Memory interface: 256-bit
- Memory bandwidth: 57.6 GB/sec
- Texture fill rate: 33.6 billion/sec



Asynchronous communications and Cuda 1.1 supported

CPU on each node: 1 processor dual-cores Intel E8200, 2.66 GHz
front side bus:1333MHz
RAM : 4Go DDR3, cache : 6Mo

1.3 - Software installed

Software	Installed & Available
• MPICH-2	yes
• OpenMPI	yes
• GCC (with OpenMP)	yes
• ICC (with OpenMP)	*no, coming soon*
• CUDA 1.1	yes
• OAR	*no, coming soon*
• Linux – Fedora core 8 (64 bit kernel)	yes
• Other software can be installed	

To support various experiments

→ Contact us

1.4 – Interconnection networks

Networks	**Installed & available**
Gigabit Ethernet	yes
Infiniband	yes, half of the cluster
10-Gigabit Ethernet	*no, next year*

GPUs compute very fast:

→ network communication times are not negligible

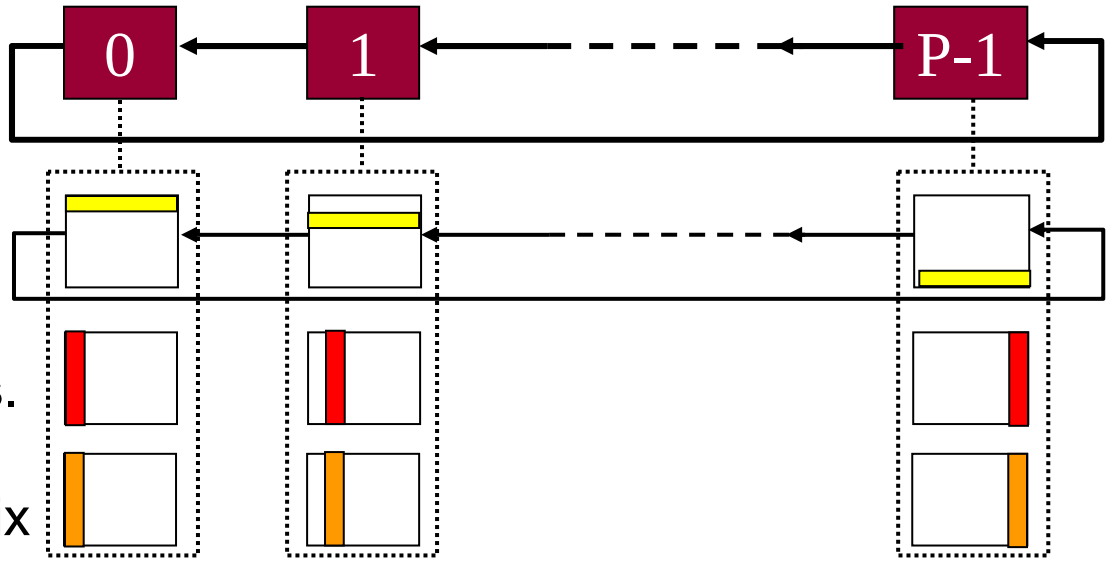
→ experiment and identify the best / less worst network

2 – First benchmarks on a GPU cluster…

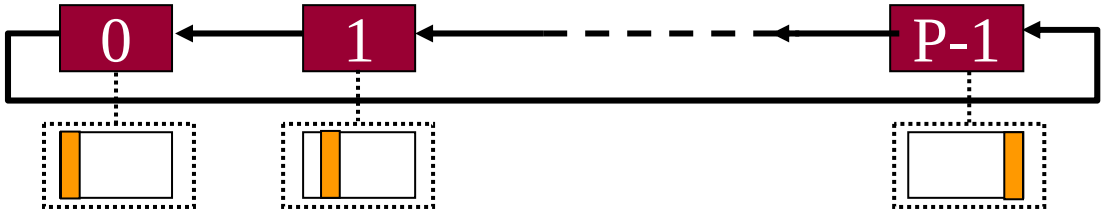
Distributed matrix product

Principles (C = AxB) :

1. Matrixes *A* and *B* are partitioned on *P* PCs.
2. The *B* partition is static.
3. The *A* partition circulates on the ring of PCs.
4. Algorithm includes *P* steps.
5. At each step, each PC computes a part of *C* matrix



6. At the end, the *C* = *AxB* matrix is distributed on the *P* PCs



Each local computations is run on the GPU…

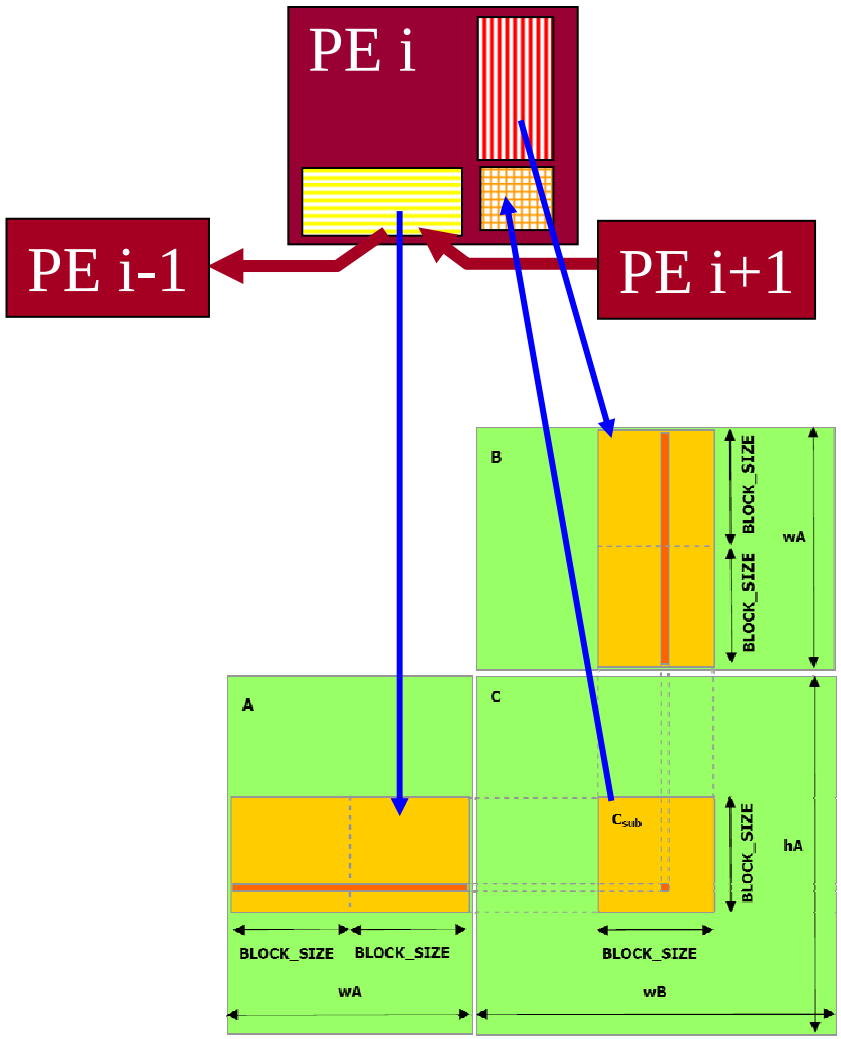
Distributed matrix product

One step on PE i:

Circulation of A partition:

CPU-GPU data transfers:

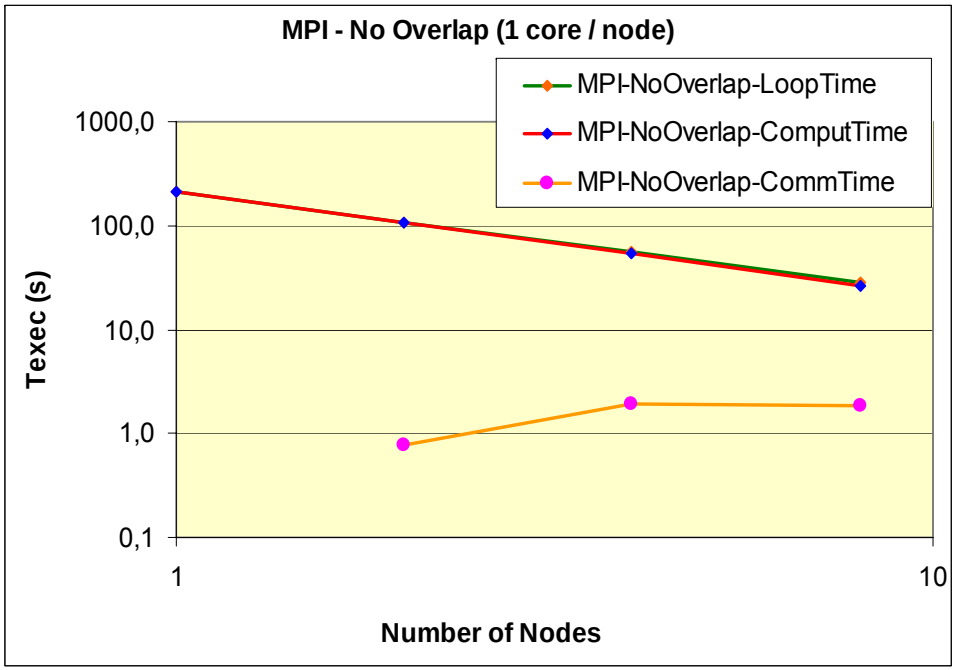
Computation of C on GPU:

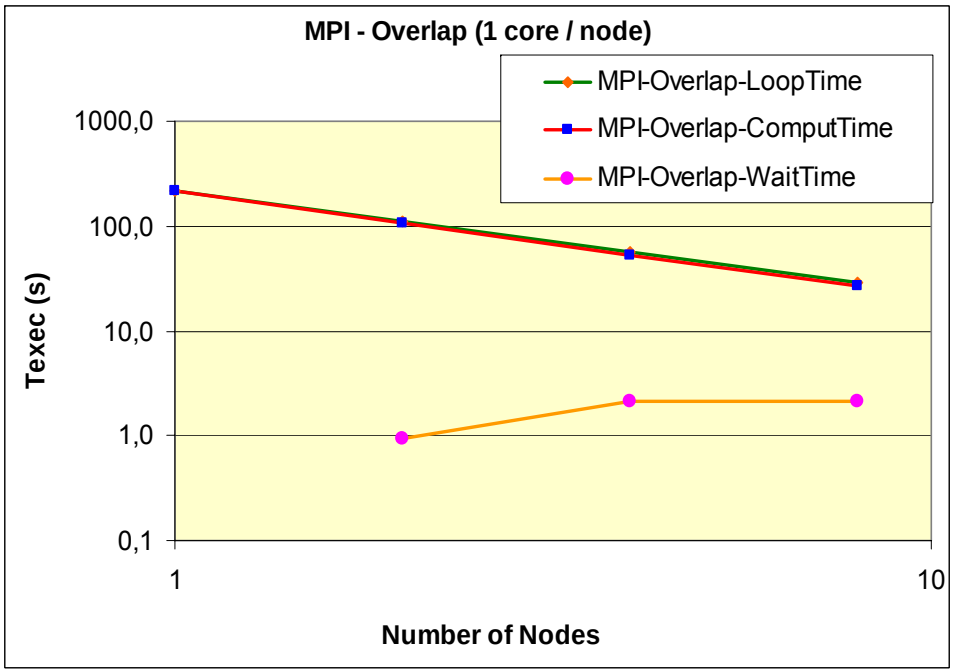


Distributed matrix product

MPI on cluster of CPUs: 1 core/node + Gigabit Ethernet

- Computation time >> communications time
  → overlapping has no impact
- Regular decrease of the execution time (good scalability)

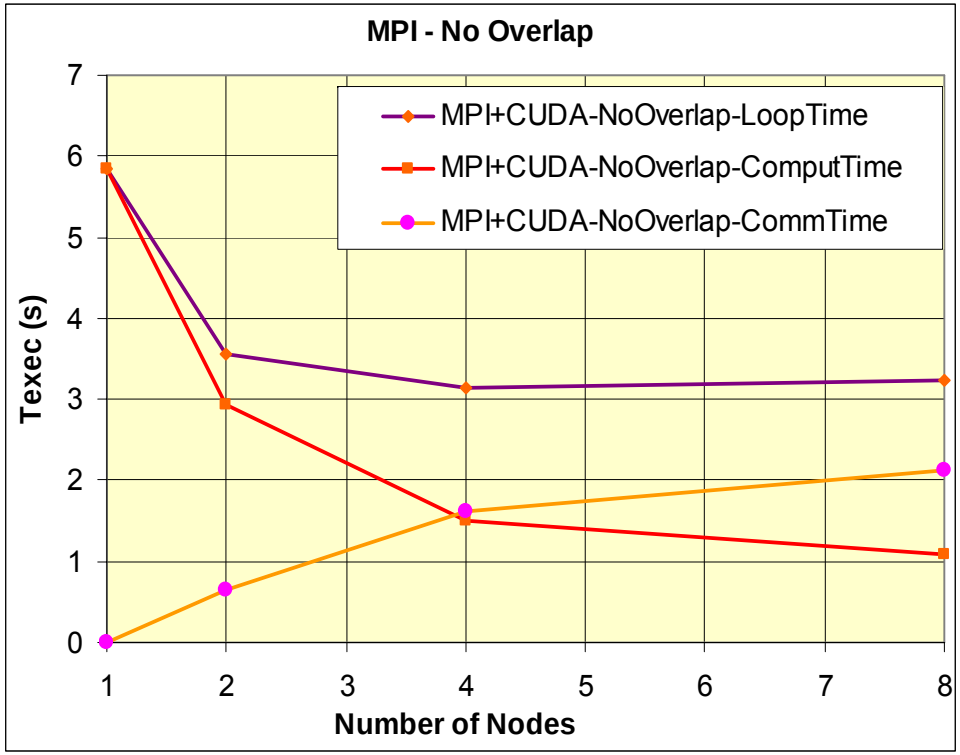


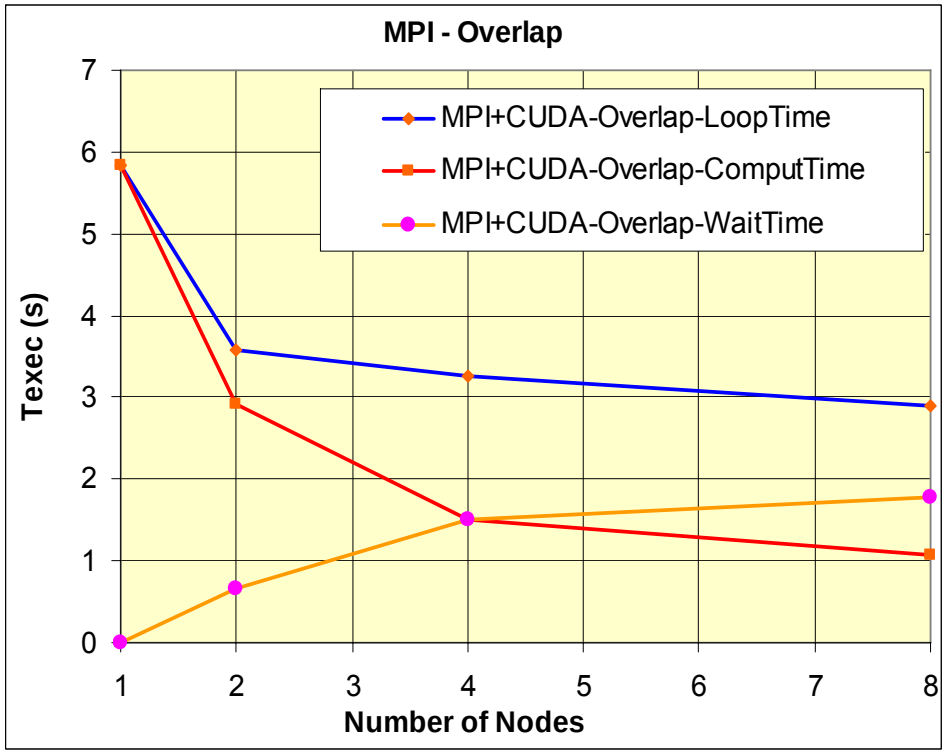


Distributed matrix product

MPI+CUDA on cluster of CPU-GPUs: 1 core/node + 1 GPU/node + Gigabit Eth

- Computation time ≈ communications time !
  → Gigabit Ethernet is not fast enough for GPU communications !
- Overlapping of CPU comms & GPU computation has an impact:
  → The overlap is incomplete, but seems the right strategy





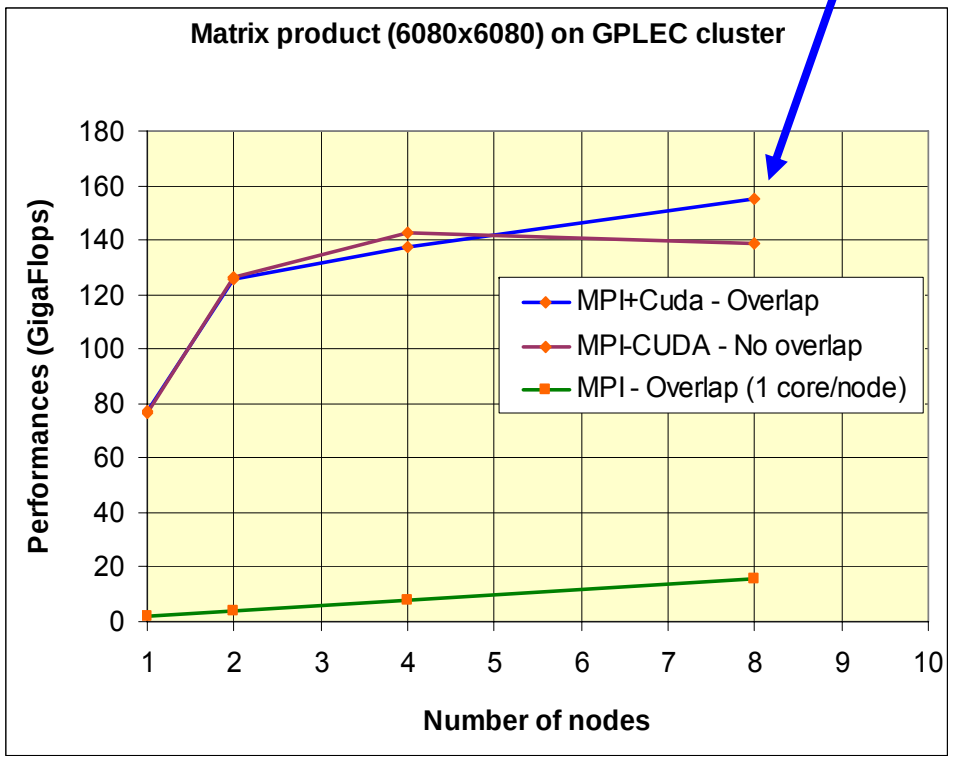
Distributed matrix product

Finally:

- 2.1 Gflops on 1 CPU
  → 155 GFlops on 8 GPUs
- But many time spent in cluster communications
- Network is slow and overlap is incomplete!
- Infiniband interconnect does not improve performances (difference appears for a larger number of nodes).

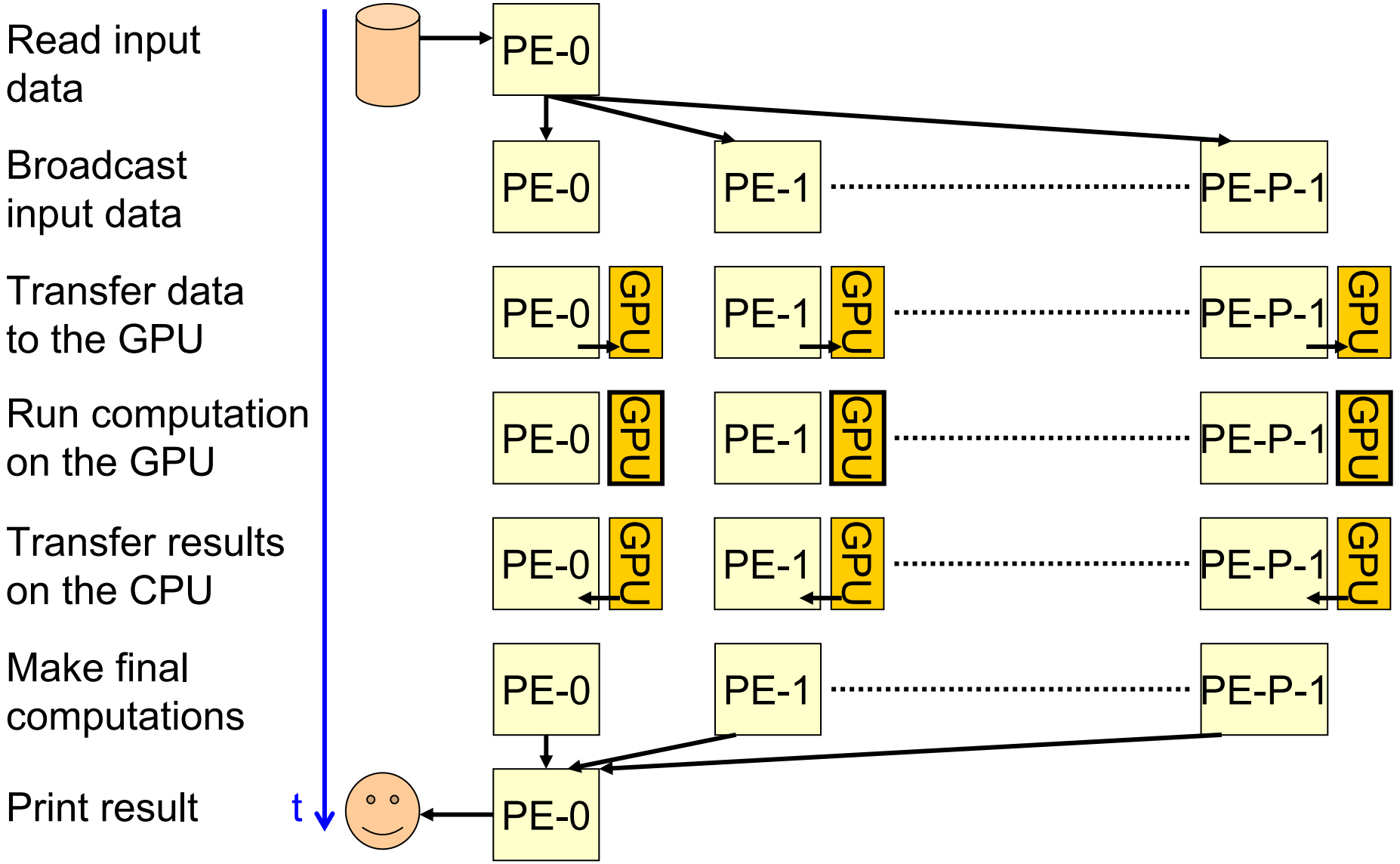
Result are encouraging but are far from peak performances!

Nb of Nodes	Nb of CPUs (1 core/node)	Nb of GPUs (1 GPU/node)
1	2.1 GFlops	77 Gflops
8	15.5 GFlops	**155 Gflops**



3 – Parallelization of an « Asian Option Pricer » on a GPU cluster

Parallelization principle



Implementation (1)

```
int main(int argc, char **argv)
{
 ... // Variable declarations

 MPI_Init(&argc, &argv);
 MPI_Comm_size(MPI_COMM_WORLD,&NbPE);
 MPI_Comm_rank(MPI_COMM_WORLD,&Me);

 if (Me == 0) {
   InitStockParaCPU();
 }
 BroadcastInputData();
 InitStockParaGPU();
 ……
```

MPI initializations.

Input data file reading from PE 0.

Broadcast input data to all PEs.

Transfer input data on GPU.

Implementation (2)

```
……
for (int jj = 0; jj <= N; jj++){
  for (int k = 0; k < NbStocks; k++){
    ComputeUniformRandom();
    GaussPRNG(k);
    OutputInputPRNG();
  }
  ActStock(jj);
  for (int k = 0; k < NbStocks; k++){
    AsianSum(k,jj);
    OutputInputSum(k);
  }
}
ComputeIntegralSum();
ComputePriceSum();

for (int i = 0; i < Nx; i++) {
  for (int j = 0; j < Ny; j++) {
    value = maxi((float)(BasketPriceCPU[i][j]-
                         BasketSumCPU[i][j]),0);
    sum = sum+value;
    sum2 = sum2+value*value;
  }
}
……
```

Call « kernels » on GPU, and transfer results on CPU.

CPU computations.

Implementation (3)

```
 ……
 MPI_Reduce(&sum,&TotalSum,1,MPI_DOUBLE,MPI_SUM,
            0,MPI_COMM_WORLD);
 MPI_Reduce(&sum2,&TotalSum2,1,MPI_DOUBLE,MPI_SUM,
            0,MPI_COMM_WORLD);

 if (Me == 0) {
   value = exp(-r)*(TotalSum/(((double)Nx)*Ny*NbPE));
   fprintf(stdout,"Computed price: %f\n",
           (float)value);
 }

 MPI_Finalize();
 return(EXIT_SUCCESS);
}
```

Collect all results on PE-0.

Compute last result on PE0.

Close MPI mechanisms

Compilation

OpenMPI + Cuda:

```
nvcc -O3                          // Serial automatic optimizations
     -I/opt/openmpi/include/      // MPI include files
     -I/usr/include/c++/4.1.2/    // Include files required by MPI
     -DOMPI_SKIP_MPICXX           // NVCC does not support « exceptions »
     -o AsianPricer
     *.cu                         // ALL source files are .cu files
```

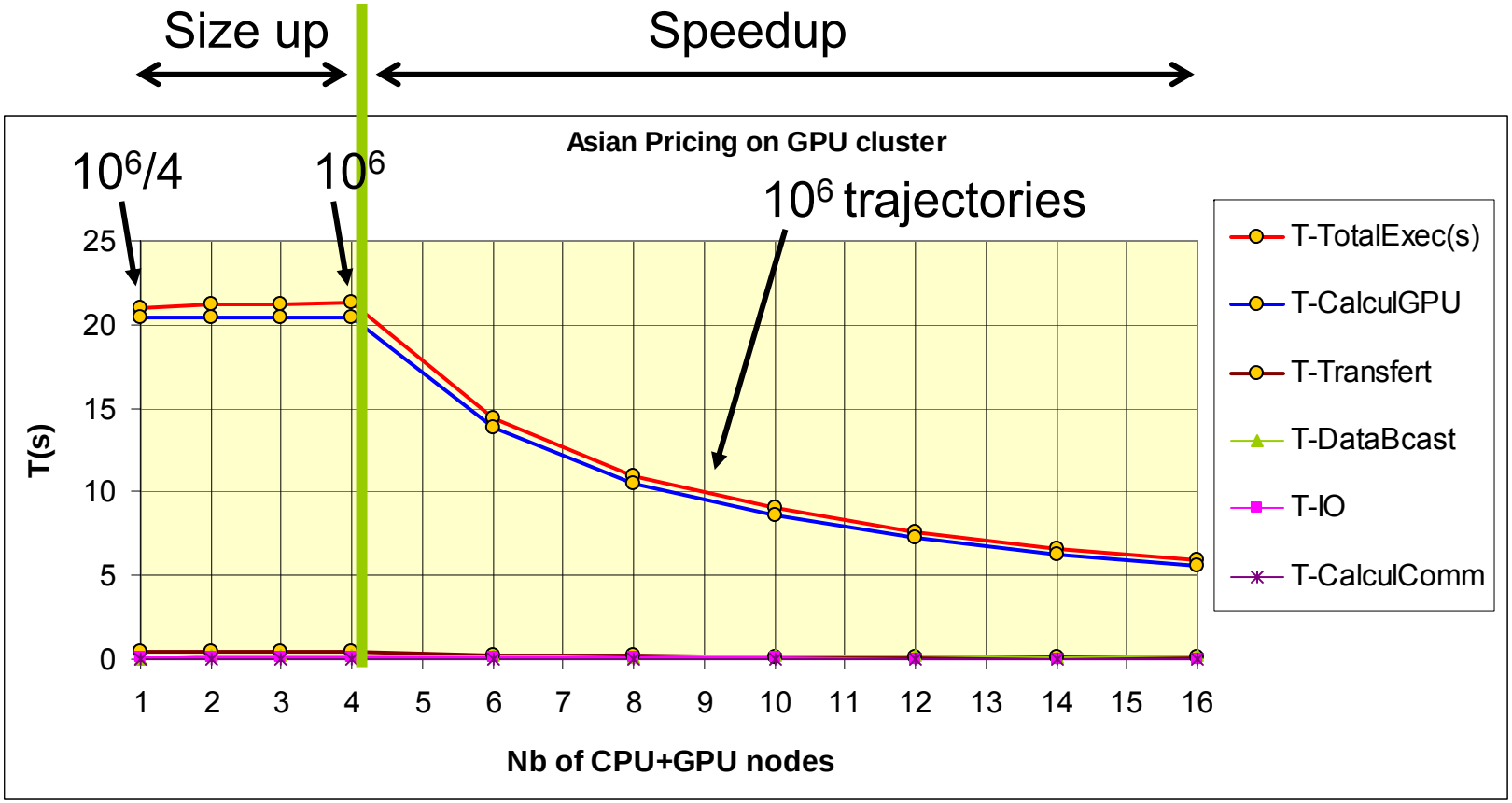
→ Compilation using OpenMPI+CUDA appears easy when all files have .cu extension

4 – Usage and performances of an « Asian Option Pricer » on a GPU cluster

Experimental performances

From 1 to 4 nodes: size up (to achieve accuracy of 10^6 trajectories),

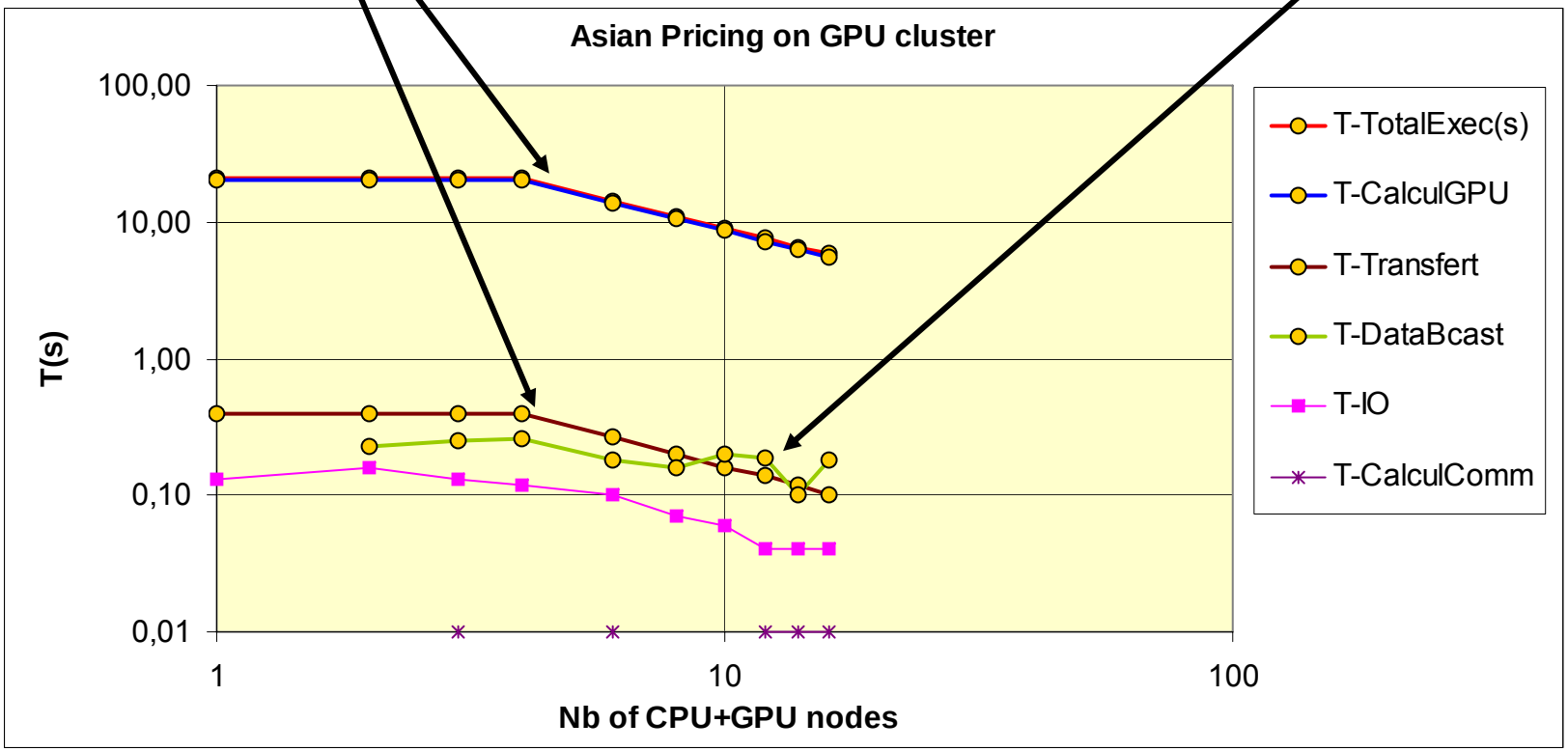
Beyond 4 nodes: speedup (to achieve computations faster).



Experimental performances

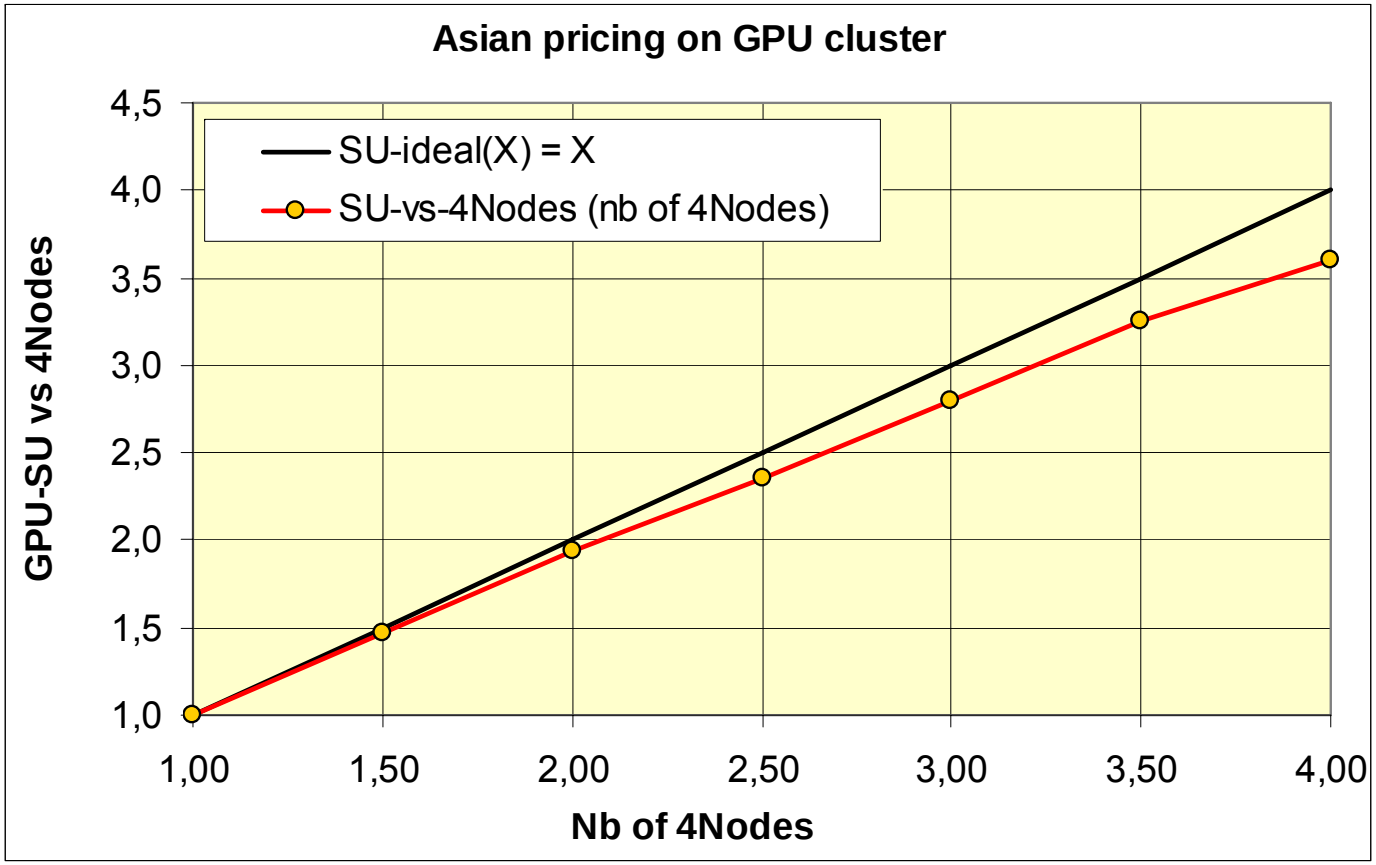
Good scaling of GPU computations and CPU-GPU transfers …

… but data broadcast could become a problem.



Experimental speedup

The « speedup » part of the experiment (from 4 nodes to 16 nodes) exhibit correct relative speedup (compared to execution on 4 nodes).



Experimental speedup … to do!

The « speedup » of the GPU cluster compared to an execution on 1 CPU & 1 core is: … ???

- Requires a sequential execution on the CPU of node of the GPU cluster
  … TO DO !
  … Previously measured close to 100 on other systems.

  → The GPU cluster could achieve a speedup close to 360, compared to a sequential execution on one CPU of the same cluster.

The « speedup » of the GPU cluster compared to an execution on a cluster of P multi-core CPUs is … ???

- Requires a parallel MPI+OpenMP execution on the CPUs of the GPU cluster
  … TO DO !

5 – Conclusion and perspectives

5 – Conclusion and perspectives

- Current results are promising.
- Size up + Speedup seems the realistic way to use a cluster of GPUs.

Future work:

- Optimize parallel algorithms and source code (many issues to investigate in MPI+CUDA programming).
- Measure performances on a cluster of multi-core CPUs, and compare.
- Measure the energy consumed and compare CPU and GPU energetic performances.

Next events:

- 2nd JTE-GPGPU, (December 4, 2008, Paris)
- PDCoF'09 (May 2009, Rome, Italy)

Supélec

région lorraine

Asian Option Pricing on cluster of GPUs: First Results

(ANR project « GCPMF »)

Questions ?