

Par4All: Auto-Parallelizing C and Fortran for the CUDA Architecture

Christopher D. Carothers⁴ Fabien Coelho² Béatrice Creusillet¹ Laurent Daverio² Stéphanie Even³ Johan Gall^{1,3} Serge Guetlon^{2,3} François Irigoien² **Ronan Keryell**^{1,3} Pierre Villalon¹

¹HPC Project

²Mines ParisTech/CRI

³Institut TELECOM/TELECOM Bretagne/HPCAS

⁴Rensselaer Polytechnic Institute/CS

10/1/2009

<http://www.par4all.org>



Par4All in CUDA — GPU conference 10/1/2009

HPC Project, Mines ParisTech, TELECOM Bretagne, PPI

Ronan KERYELL et al.

1 / 46

• Introduction

HPC Project software and services

- Parallelize and optimize customer applications, co-branded as a bundle product in a WildNode (e.g. Presagis Stage battle-field simulator)
- Acceleration software for the WildNode
 - ▶ GPU-accelerated libraries for Matlab/Octave/R
 - ▶ Transparent execution of Matlab on the WildNode
- Remote display software for Windows on the WildNode

HPC consulting

- ▶ Optimization and parallelization of applications
- ▶ *High Performance?*... not only TOP500-class systems: power-efficiency, embedded systems, green computing...
- ▶ Embedded system and application design
- ▶ Training in parallel programming (OpenMP, MPI, TBB, CUDA, OpenCL...)



Par4All in CUDA — GPU conference 10/1/2009

HPC Project, Mines ParisTech, TELECOM Bretagne, PPI

Ronan KERYELL et al.

3 / 46

• Introduction

HPC Project hardware: WildNode from Wild Systems

Through its Wild Systems subsidiary company

- WildNode hardware desktop accelerator
 - ▶ Low noise for in-office operation
 - ▶ x86 manycore
 - ▶ nVidia Tesla GPU Computing
 - ▶ Linux & Windows



• WildHive

- ▶ Aggregate 2-4 nodes with 2 possible memory views
 - Distributed memory with Ethernet or InfiniBand
 - Virtual shared memory through Linux Kernighed for single-image system

<http://www.wild-systems.com>

Par4All in CUDA — GPU conference 10/1/2009

HPC Project, Mines ParisTech, TELECOM Bretagne, PPI

Ronan KERYELL et al.

2 / 46



• Introduction

We need software tools

- HPC Project needs tools for its hardware accelerators (*Wild Nodes* from *Wild Systems*) and to parallelize, port & optimize customer applications
- Application development: long-term business ~ long-term commitment in a tool that needs to survive to (too fast) technology change
- Unreasonable to begin yet another new compiler project...
- Many academic Open Source projects are available...
- ...But customers need products ☺

Par4All

- Funding an initiative to industrialize Open Source tools
 - PIPS is the first project to enter the Par4All initiative
- <http://www.par4all.org>



Par4All in CUDA — GPU conference 10/1/2009

HPC Project, Mines ParisTech, TELECOM Bretagne, PPI

Ronan KERYELL et al.

4 / 46

Outline

- 1 Par4All
- 2 CUDA generation
- 3 Results
- 4 Conclusion

Par4All in CUDA — GPU conference 10/1/2009
HPC Project, Mines ParisTech, T EL ECOM Bretagne, RPI

Ronan KERVILL et al.

5 / 46



PIPS

(I)

- PIPS (Interprocedural Parallelizer of Scientific Programs): Open Source project from Mines ParisTech... \approx 150 hy, 21-year old!
- Funded by many people (French DoD, Industry & Research Departments, University, CEA, IFR, Onera, ANR (French NSF), European projects, regional research clusters...)
- Project that coined polytype model-based compilation
- \approx 456 KLOC according to David A. Wheeler's SLUCCount
- ... but modular and sensible approach to pass through the years
 - ▶ \approx 300 phases (parsers, analyzers, transformations, optimizers, parallelizers, code generators, pretty-printers...) that can be combined for the right purpose
 - ▶ NewGen object description language for language-agnostic automatic generation of methods, persistence, object introspection, visitors, accessors, constructors, XML marshaling for interfacing with external tools...

Par4All in CUDA — GPU conference 10/1/2009
HPC Project, Mines ParisTech, T EL ECOM Bretagne, RPI

Ronan KERVILL et al.

7 / 46



Use the Source, Luke...

Hardware is moving quite (too) fast but...

What has survived for 50+ years?

Fortran programs...

What has survived for 30+ years?

C programs...

- A lot of legacy code could be pushed onto parallel hardware (accelerators) with automatic tools...
- Not as efficient as hand-tuned programs, but quick production phase
- Need automatic tools for source-to-source transformation to leverage existing software tools for a given hardware

Par4All in CUDA — GPU conference 10/1/2009
HPC Project, Mines ParisTech, T EL ECOM Bretagne, RPI

Ronan KERVILL et al.

6 / 46



PIPS

(II)

- ▶ Interprocedural * a la* make engine to chain the phases as needed.
 - ▶ Lazy construction of resources
 - ▶ Polytype lattice (linear algebra) used for semantics analysis, transformations, code generation... to deal with big programs, not only loop-nests
 - ▶ Huge on-going efforts to industrialize the project, extension of the semantics analysis for C
- Around 15 programmers currently developing in PIPS (Mines ParisTech, HPC Project, IT SudParis, T EL ECOM Bretagne, RPI) with public svn, Trac, mailing lists, IRC, Plone, Skype... and use it for many projects
- But still...
 - ▶ Huge need of documentation (even if PIPS uses literate programming...)
 - ▶ Need of industrialization
 - ▶ Need further communication to increase community size

Par4All in CUDA — GPU conference 10/1/2009
HPC Project, Mines ParisTech, T EL ECOM Bretagne, RPI

Ronan KERVILL et al.

8 / 46



Current PIPS usage

- Automatic parallelization (C & Fortran to OpenMP)
- Distributed memory computing with OpenMP-to-MPI translation (STEP project)
- Generic vectorization for SIMD instructions (SSE, VMX...) (SAC project)
- Parallelization for embedded systems (SCALOPES)
- Compilation for hardware accelerators (Ter@PIX, SPoC, SIMD, FPGA...)
- High-level hardware accelerators synthesis generation for FPGA
- Reverse engineering & decompiler (reconstruction from binary to C)

Logical next step

GPU! ☺



Basic CUDA execution model

A sequential program on a host launches computational-intensive kernels on a GPU

- Allocate storage on the GPU
- Copy-in data from the host to the GPU
- Launch the kernel on the GPU
- Copy-out the results from the GPU to the host
- Deallocate the storage on the GPU



Outline

- 1 Par4All
- 2 CUDA generation
- 3 Results
- 4 Conclusion



Challenges in automatic CUDA generation

- Find parallel kernels
- Improve data reuse inside kernels to have better compute intensity (even if the memory bandwidth is quite higher than on a CPU...)
- Access the memory in a GPU-friendly way (to coalesce memory accesses)
- Take advantage of complex memory hierarchy that make the GPU fast (shared memory, cached texture memory, registers...)
- Reduce the copy-in and copy-out transfers that pile up on the PCIe
- Reduce memory usage in the GPU (no swap there, yet...)
- Limit inter-block synchronizations
- Overlap computations and GPU-CPU transfers (via streams)




Automatic parallelization

Most fundamental for a parallel execution

Finding parallelism!

Several parallelization algorithms are available in PIPS

- For example classical Allen & Kennedy use loop distribution more vector-oriented than kernel-oriented  (or need later loop-fusion)
- Coarse grain parallelization based on the independence of array regions used by different loop iterations
 - ▶ Currently used because generates GPU-friendly coarse-grain parallelism
 - ▶ Accept complex control code without *if-conversion*



Outlining

• After:

```

1 p4a_kernel_launcher_0(space, save);
2 [...]
3 void p4a_kernel_launcher_0(float_t space[SIZE],
4     float_t save[SIZE]) {
5     for (i = 1; i <= 499; i += 1)
6         for (j = 1; j <= 499; j += 1)
7             p4a_kernel_0(i, j, save, space);
8 }
9 [...]
10 void p4a_kernel_0(float_t space[SIZE],
11     float_t save[SIZE]) {
12     int i,
13         save[i][j] = 0.25*(space[i-1][j]+space[i+1][j]
14     +space[i][j-1]+space[i][j+1]);
15 }

```

(II)



Outlining

Parallel code  Kernel code on GPU

- Need to extract parallel source code into kernel source code: outlining of parallel loop-nests

• Before:

```

1 for (i = 1; i <= 499; i++)
2     for (j = 1; j <= 499; j++) {
3         save[i][j] = 0.25*(space[i - 1][j] + space[i + 1][j]
4         + space[i][j - 1] + space[i][j + 1]);
5     }

```



From array regions to GPU memory allocation (I)

- Memory accesses are summed up for each statement as *regions* for array accesses: integer polytope lattice
- There are regions for write access and regions for read access
- The regions can be **exact** if PIPS can **prove** that **only** these points are accessed, or they can be **inexact**, if PIPS can only find an over-approximation of what is really accessed



From array regions to GPU memory allocation (II)

Example

```

1  for(i = 0; i <= n-1; i += 1)
2  for(j = 1; j <= n-1; j += 1)
   h_A[i][j] = 1;

```

can be decorated by PIPS with write array regions as:

```

1  // <h_A[PHI1]//PHI2]-WEXACT- {0<=PHI1, PHI2+k<=n, PHI1<=PHI2}>
   for(i = 0; i <= n-1; i += 1)
2  // <h_A[PHI1]//PHI2]-WEXACT- {PHI1=i, i<=PHI2, PHI2+k<=n, 0<=i}>
   for(j = 1; j <= n-1; j += 1)
3  // <h_A[PHI1]//PHI2]-WEXACT- {PHI1=i, PHI2=j, 0<=i, i<=j, 1+j<=n}>
   h_A[i][j] = 1;

```

- These read/write regions for a kernel are used to allocate with a `cudaMalloc()` in the host code the memory used inside a kernel and to deallocate it later with a `cudaFree()`



Communication generation (II)

- PIPS gives 2 very interesting region types for this purpose
 - ▶ **In-region** abstracts what really needed by a statement
 - ▶ **Out-region** abstracts what really produced by a statement to be used later elsewhere
- In-Out regions can directly be translated with CUDA into
 - ▶ **copy-in**

```

1  cudaMemcpy (accel_address, host_address,
2             size, cudaMemcpyHostToDevice)

```
 - ▶ **copy-out**

```

1  cudaMemcpy (host_address, accel_address,
2             size, cudaMemcpyDeviceToHost)

```



Communication generation (I)

Conservative approach to generate communications

- Associate any GPU memory allocation with a copy-in to keep its value in sync with the host code
- Associate any GPU memory deallocation with a copy-out to keep the host code in sync with the updated values on the GPU

- But a kernel could initialize an array, or use the initial values without writing into it, or use it as a local (private) array
- (PIPS does have many privatization phases)



Loop normalization

- Hardware accelerators use fixed iteration space (CUDA: thread index starting from 0..)
- Parallel loops: more general iteration space
- Loop normalization

```

Before
1  for(i = 1; i < SIZE - 1; i++)
2  for(j = 1; j < SIZE - 1; j++) {
   save[i][j] = 0.25*(space[i - 1][j] + space[i + 1][j]
3             + space[i][j - 1] + space[i][j + 1]);
4  }

```

```

After
1  for(i = 0; i < SIZE - 2; i++)
   for(j = 0; j < SIZE - 2; j++) {
2  save[i+1][j+1] = 0.25*(space[i][j + 1] + space[i + 2][j + 1]
3  + space[i+1][j+1] + space[i + 1][j]
4  + space[i + 1][j] + space[i + 1][j]
5  );

```



From preconditions to iteration clamping

(I)

- Parallel loop nests are compiled into a CUDA kernel wrapper launch
- The kernel wrapper itself gets its virtual processor index with `some blockIdx.x * blockDim.x + threadIdx.x`
- Since only full blocks of threads are executed, if the number of iterations in a given dimension is not a multiple of the `blockDim`, there are incomplete blocks
- An incomplete block means that some index overrun occurs if all the threads of the block are executed



Complexity analysis

(II)

- Launching a GPU kernel is expensive
 - ▶ so we need to launch only kernels with a significant speed-up (launching overhead, memory CPU-GPU copy overhead...)
- Some systems use `#pragma` to give a go/no-go information to parallel execution


```
1 #pragma omp parallel if (size>100)
```
- ∃ phase in PIPS to symbolically estimate complexity of statements
- Based on preconditions
- Use a SuperSparc2 model from the '90s... ☹
- Can be changed, but precise enough to have a coarse go/no-go information
- To be refined: use memory usage complexity to have information about memory reuse (even a big kernel could be more efficient on a CPU if there is a good cache use)



From preconditions to iteration clamping

(III)

- So we need to generate code such as

```
1 void p4a_kernel_wrapper_0(int k, int l, ...)
2 {
3     k = blockIdx.x * blockDim.x + threadIdx.x;
4     l = blockIdx.y * blockDim.y + threadIdx.y;
5     if (k >= 0 && k <= M - 1 && l >= 0 && l <= M - 1)
6         kernel(k, l, ...);
7 }
```

But how to insert these guards?

- The good news is that PIPS owns *preconditions* that are predicates on integer variables. Preconditions at entry of the kernel are:


```
1 // P(i, j, k, l) {0 <= k, k <= 63, 0 <= l, l <= 63}
```
- Guard ≡ directly translation in C of preconditions on loop indices that are GPU thread indices



Optimized reduction generation

(IV)

- Reduction are common patterns that need special care to be correctly parallelized
- Reduction detection already implemented in PIPS
- Efficient computation in CUDA needs to create local reduction trees in the thread-blocks
- On-going implementation at Rensselaer Polytechnic Institute (RPI) with parallel-prefix



Fortran to CUDA

(I)

- Fortran 77 parser available in PIPS
- CUDA is C/C++ with some restrictions on the GPU-executed parts
- Need a Fortran to C translator (`f2c...`)?
- Only one internal representation is used in PIPS
 - ▶ Use the Fortran parser
 - ▶ Use the C pretty-printer
- But the IO Fortran library is complex to use... and to translate
 - ▶ If you have IO instructions in a Fortran loop-nest, it is not parallelized anyway because of sequential side effects ☹
 - ▶ So keep the Fortran output everywhere but in the parallel CUDA kernels
 - ▶ Apply a memory access transposition phase `a(i,j) ~> a[j-1][i-1]` inside the kernels to be pretty-printed as C
- We could output CUDA Fortran directly when available



Fortran to CUDA

(III)

- Object-oriented Fortran 2003? Hard for a compiler to understand the object-oriented interprocedural polymorphism...
- Programmer should avoid it at low-level for performance issues
 - ▶ Some high-performance optimizations need to change data organization...
 - ▶ ... that means object model restructuring ☹
- Is there a cultural community big enough to have object-oriented Fortran programs? Need a big library ecosystem to have competitive advantage...



Fortran to CUDA

(II)

- Fortran 90 and 95 support is quite interesting
 - ▶ Dynamic allocation
 - Avoid nasty user allocation functions in Fortran 77 used to allocate objects in a big array...
 - Was a parallelism killer...
 - ▶ Derived types (C structures)
 - Avoid using more dimensions to arrays to simulate structure fields in Fortran 77
 - Was a parallelism killer, used useless dereferencing...
 - ▶ Pointers
 - Useful to navigate through recursive derived type
 - Was a parallelism killer, used useless dereferencing...
 - Mmm... Try to avoid this anyway ☹
- Most of these concepts are in C, so are already dealt by PIPS
- ~> On-going Fortran 95 support with the `gfc` parser from GCC Gfortran



Par4All accel runtime — the big picture

(I)

- CUDA can not be directly represented in the internal representation (IR, abstract syntax tree) such as `__device__` or `<<<< >>>>`
- PIPS motto: keep the IR as simple as possible
- Use some calls to intrinsics functions that can be represented directly
- Intrinsics functions are implemented with (macro-)functions
 - ▶ `p4a_accel.h` has indeed currently 2 implementations
 - `p4a_accel-cuda.h` than can be compiled with CUDA for nVidia GPU execution or emulation on CPU
 - `p4a_accel-openmp.h` that can be compiled with an OpenMP compiler for simulation on a (multicore) CPU



Par4All accel runtime — the big picture

(III)

```

1  int main(int argc, char *argv[]) {
2  [...]
3  float_t (*p4a_var_space)[SIZE][SIZE];
4  P4A_ACCEL_MALLOC(&p4a_var_space, sizeof(space));
5  P4A_COPY_T0_ACCEL(space, p4a_var_space, sizeof(space));
6
7  float_t (*p4a_var_save)[SIZE][SIZE];
8  P4A_ACCEL_MALLOC(&p4a_var_save, sizeof(save));
9  P4A_COPY_T0_ACCEL(save, p4a_var_save, sizeof(save));
10
11  P4A_ACCEL_TIMER_START;
12
13  for (t = 0; t < T; t++)
14      compute(*p4a_var_space, *p4a_var_save);
15
16  double execution_time = P4A_ACCEL_TIMER_STOP_AND_FLOAT_MEASURE();
17  P4A_COPY_FROM_ACCEL(space, p4a_var_space, sizeof(space));
18
19  P4A_ACCEL_FREE(p4a_var_space);
20  P4A_ACCEL_FREE(p4a_var_save);

```

Par4All in CUDA — GPU conference 10/1/2009

HPC Project, Mines ParisTech, TELECOM Bretagne, PPI

Rohan KERVILL et al.

29 / 46



Par4All accel runtime — the big picture

(IV)

```

41  if (i >= 1 && i <= SIZE - 1 && j >= 1 && j <= SIZE - 1)
42      p4a_kernel_0(space, save, i, j);
43  }
44  P4A_ACCEL_KERNEL void p4a_kernel_0(float_t space[SIZE][SIZE],
45                               float_t save[SIZE][SIZE],
46                               int i,
47                               int j) {
48      save[i][j] = 0.25*(space[i-1][j]+space[i+1][j]
49                      +space[i][j-1]+space[i][j+1]);
50  }

```

Par4All in CUDA — GPU conference 10/1/2009

HPC Project, Mines ParisTech, TELECOM Bretagne, PPI

Rohan KERVILL et al.

31 / 46



Par4All accel runtime — the big picture

(III)

```

21  [...]
22  }
23  void compute(float_t space[SIZE][SIZE],
24              float_t save[SIZE][SIZE]) {
25  [...]
26      p4a_kernel_launcher_0(space, save);
27  [...]
28  }
29  void p4a_kernel_launcher_0(float_t space[SIZE][SIZE],
30                             float_t save[SIZE][SIZE]) {
31      P4A_CALL_ACCEL_KERNEL_2D(p4a_kernel_wrapper_0, SIZE, SIZE,
32                               space, save);
33  }
34  P4A_ACCEL_KERNEL_WRAPPER void
35  p4a_kernel_wrapper_0(float_t space[SIZE][SIZE],
36                      float_t save[SIZE][SIZE]) {
37      int j;
38      int i;
39      i = P4A_VP_0;
40      j = P4A_VP_1;

```

Par4All in CUDA — GPU conference 10/1/2009

HPC Project, Mines ParisTech, TELECOM Bretagne, PPI

Rohan KERVILL et al.

30 / 46



Outline

- 1 Par4All
- 2 CUDA generation
- 3 Results
- 4 Conclusion

Par4All in CUDA — GPU conference 10/1/2009

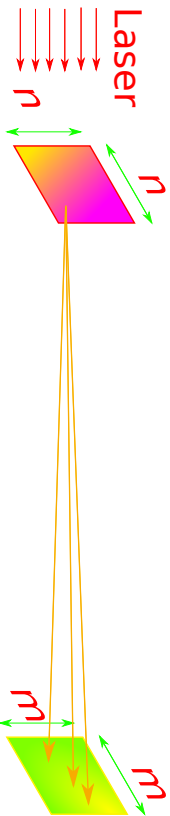
HPC Project, Mines ParisTech, TELECOM Bretagne, PPI

Rohan KERVILL et al.

32 / 46



Results on a customer application



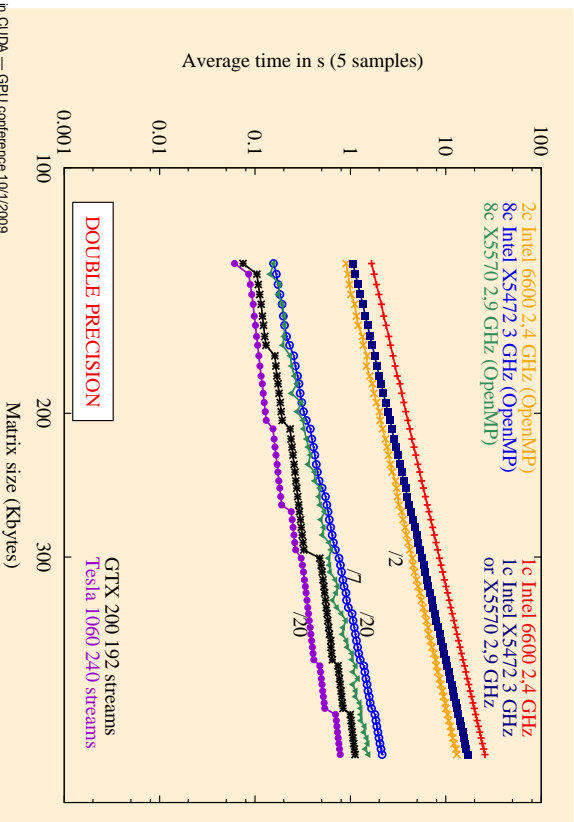
- Holotetrix's primary activities are the design, fabrication and commercialization of prototype diffractive optical elements (DOE) and micro-optics for diverse industrial applications such as LED illumination, laser beam shaping, wavefront analyzers, etc.
- Hologram verification with direct Fresnel simulation
- Program in C
- Parallelized with
 - ▶ Par4All CUDA and CUDA 2.3, Linux Ubuntu x86-64
 - ▶ Par4All OpenMP, gcc 4.3, Linux Ubuntu x86-64
- Reference: Intel Core2 6600 @ 2.40GHz

<http://www.holotetrix.com>

Visit us at the Wild Systems booth #35 for a demo

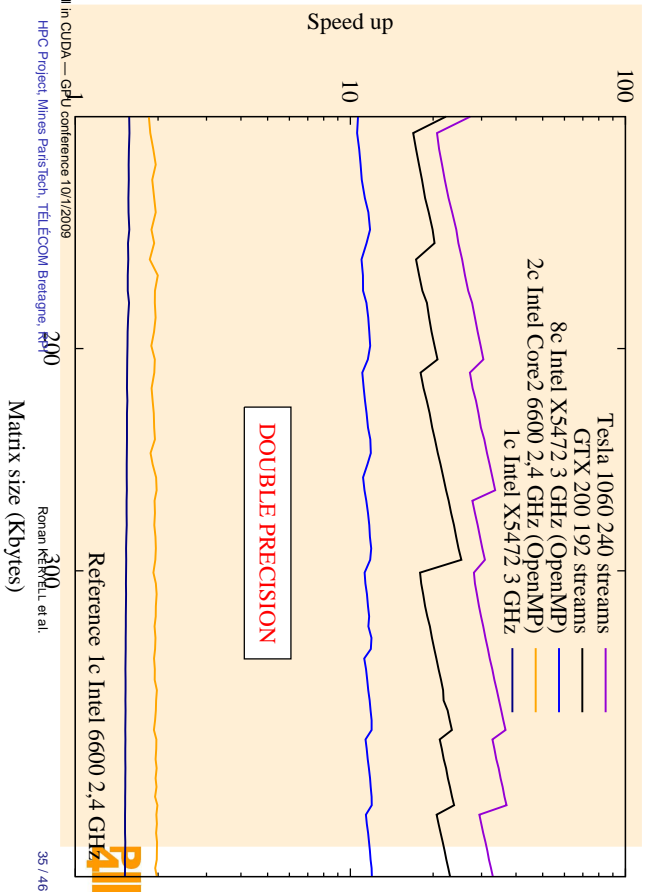


Comparative performance



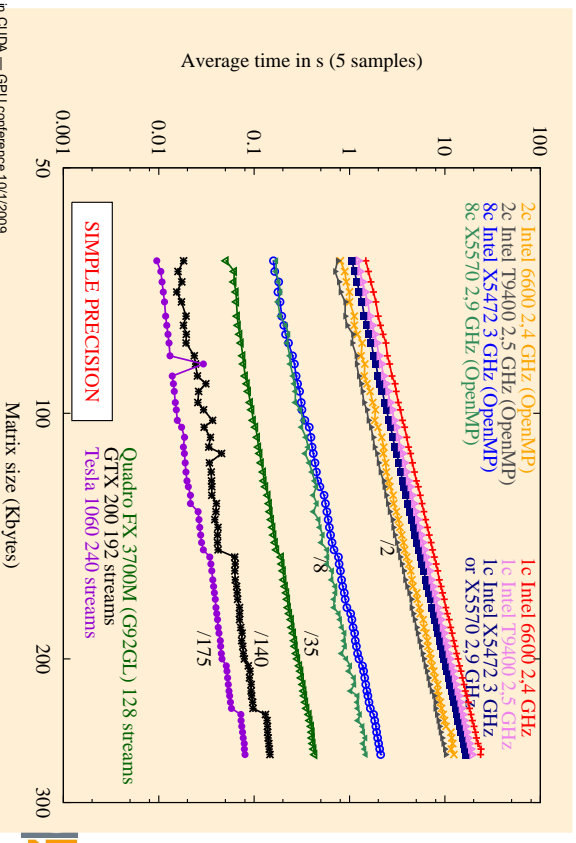
Comparative performance

(11)



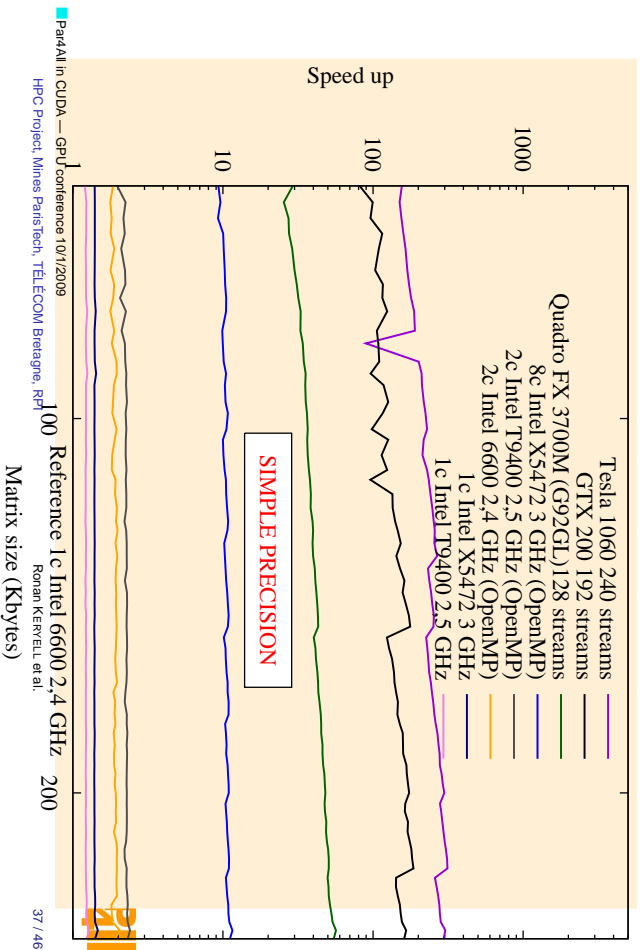
Keep it simple (precision)

(1)



Keep it simple (precision)

(II)



Take advantage of C99

(I)

- BaC99k to C99imple C99ings
- Avoiding using C99 umbersome old C C99onstruC99ts C99an lead to C99learner C99ode, more effiC99ient and more parallelizable C99ode with Par4All
- C999 adds C999ympaC999etiC999 features that are unfortunately not well known:
 - ▶ MultidimenC999ional arrays with non-staticC999ec size
 - Avoid malloc() and useless pointer C999onstruC999ions
 - You C999an have arrays with a dynamicC999 size in funC999tion parameters (as in Fortran)
 - Avoid useless linearizing C999omputaC999ions (a[i][j] instead of a[i+h*j] ...)
 - Avoid most of alloca()
 - ▶ TLS (Thread-Local Storage) ExtenC999ion C999an express independent storage
 - ▶ C999omplex numbers and boolens avoid further struC999tures or enums



Outline

- 1 Par4All
- 2 CUDA generation
- 3 Results
- 4 Conclusion



From an open source project to genetic algorithms

Why open source?

- Once upon a time, a research team in Luxembourg decided to experiment program transformations with genetic algorithms...
- Main idea:
 - ▶ Represent the transformations applied to a program as a gene
 - ▶ Make the genes to evolve
 - ▶ Measure the performance
 - ▶ Select and iterate
- They used PIPS to implement this
- Side effects
 - ▶ Now PIPS is scriptable in Python (PYPS)
 - ▶ Algorithm written in Python
 - ▶ We have an automated way to try many different optimizations to generate better CUDA kernels!

Stay tuned...



Future work

(1)

- Use advanced tiling to take advantage of shared memory and texture cache
- Select optimal transformations and code generation parameter by using genetic algorithm system already in PIPS
- Using complexity information to select the best execution mode (GPU or CPU)
- Mix GPU and CPU execution, with SSEx, OpenMP and/or MPI
- Matlab to CUDA compiler
- OpenCL support via OpenGPU project
- Eclipse (EclIPPS) integration via OpenGPU project
- If someone spent some time to write #pragma information... Use them! ☺
- Generate run-time specialization and speculation when some static information is lacking at compilation time



Conclusion

(1)

- GPU (and other heterogeneous accelerators): impressive peak performances and memory bandwidth, power efficient
- Looks like to the 80's : many accelerators, vectors, parallel computers
 - ▶ Need to port the programs to new architectures
 - ▶ Many start-ups
- ...but some hypothesis changed, hardware constraints ~ we can no longer escape parallelism! No longer a niche market!
- Programming is *quite more* complex... ☹
- Non-functional specifications (speed, time, power, parallelism...) not taken into account by main-stream programming environment and teaching (high-level object programming...)
- And programs are quite bigger now! ☹
- Application source code evolves slowly compared to hardware



Future work

(1)

- We have already PhD students, engineers and researchers around Par4All, but many hard issues remains, so we need you too... ☺



Conclusion

(1)

- Capitalize on this with a source-to-source tool!
- Before addressing parallelism, may be interesting to optimize the sequential program (algorithmic, coding, right tools...)
- Take a positive attitude... Parallelization is a good opportunity for deep cleaning (refactoring, modernization...) ~ improve also the original code
- A cleaner code is often easier to parallelize automatically or manually
- Do not forget simple precision exist too... ☹
- ↕ Entry cost
- ↕ Exit cost! ☹
- Par4All initiative aims at minimizing entry cost and exit cost (CUDA for nVidia GPU)
 - ▶ Use source-to-source compilers to surf on the best back-ends
 - ▶ Avoid manual work with preprocessing, tools, code generators



Conclusion

(III)

- ▶ Open source architecture to collaborate with best research teams
- ▶ Use standards and Open Source for long-term investment
- ▶ Open source to avoid sticking to a one-company solution
- ▶ Hide manual work in preprocessing, tools, code generators...
- ▶ Can accept `#pragma` from other products as a second-source
- ▶ Come on and participate to the initiative!

You are not a compiler developer and you may ask what to keep from such a talk?

If you **understand better** how a compiler work, you can write code that **fits better** the compiler... ☺

Visit us at the Wild Systems booth #35 for a demo



Par4All in CUDA — GPU conference 10/1/2009
HPC Project, Mines ParisTech, TELECOM Bretagne, PPI

Ronan KERVELL et al.

46 / 46

• Table des matières

HPC Project hardware, WildNode from Wild Systems	21
HPC Project software and services	23
We need software tools	24
1 Par4All	25
Outline	28
Use the Source, Like...	32
PIPS	33
Current PIPS usage	34
2 CUDA generation	36
Outline	38
Basic CUDA execution model	39
Challenges in automatic CUDA generation	40
Automatic parallelization	41
Outlining	43
From array regions to GPU memory allocation	46
Communication generation	47
Loop normalization	
3 Results	32
Outline	33
Results on a customer application	34
Comparative performance	34
Keep it simple (precision)	36
4 Conclusion	38
Outline	39
Take advantage of C99	39
From an open source project to generic algorithms	40
Future work	41
Conclusion	43
Par4All is currently supported by...	46
You are here!	47

Par4All is currently supported by...

- HPC Project
- Mines ParisTech
- Institut TELECOM/TELECOM Bretagne
- Rensselaer Polytechnic Institute
- European ARTEMIS SCALOPES project
- French NSF (ANR) FREIA project
- French Images and Networks research cluster TransMedi@project
- French System@TIC research cluster OpenGPU project



Par4All in CUDA — GPU conference 10/1/2009
HPC Project, Mines ParisTech, TELECOM Bretagne, PPI

Ronan KERVELL et al.

46 / 46

