

Par4All

Open source parallelization for heterogeneous computing

OpenCL & more

Ronan KERYELL³ (rk@hpc-project.com)

HPC Project

9 Route du Colonel Marcel Moraine¹
92360 Meudon La Forêt, France

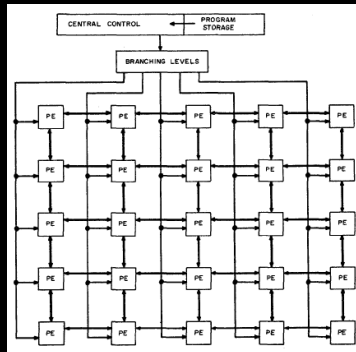
Rond Point Benjamin Franklin²
34000 Montpellier, France

Wild Systems, Inc.
5201 Great America Parkway #3241³
Santa Clara, CA 95054, USA

24/01/2012

SOLOMON

- Target application: “data reduction, communication, character recognition, optimization, guidance and control, orbit calculations, hydrodynamics, heat flow, diffusion, radar data processing, and numerical weather forecasting”

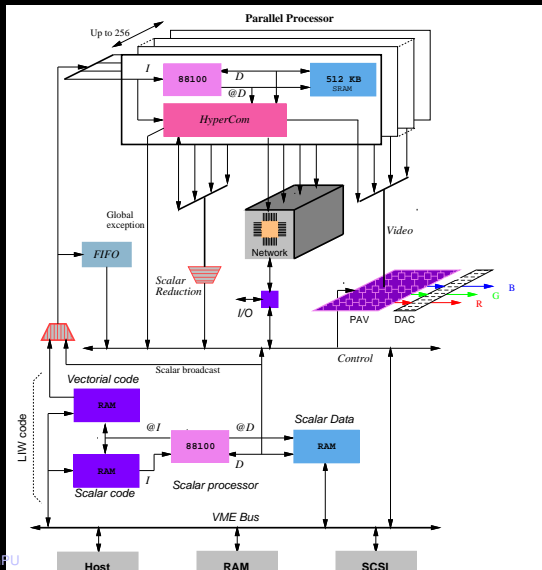


Daniel L. Slotnick. « The SOLOMON computer. »
Proceedings of the December 4-6, 1962, fall joint computer conference. p. 97–107. 1962

- Diode + transistor logic in 10-pin TO5 package



POMP & PompC @ LI/ENS 1987–1992



HyperParallel Technologies (1992–1998)

- Parallel computer
- Proprietary 3D-torus network
- DEC Alpha 21064 + FPGA
- HyperC (follow-up of PompC @ LI/ENS Ulm)
 - ▶ PGAS (Partitioned Global Address Space) language
 - ▶ An ancestor of UPC...
- Already on the Saclay Plateau ! ☺

Quite simple business model

- Customers need just to rewrite all their code in HyperC ☺
- Difficult entry cost... ☹
- Niche market... ☹
- American subsidiary with dataparallel datamining application acquired by Yahoo! in 1998
- Closed technology → lost for customers and... founders ☹

HyperParallel Technologies (1992–1998)

- Parallel computer
- Proprietary 3D-torus network
- DEC Alpha 21064 + FPGA
- HyperC (follow-up of PompC @ LI/ENS Ulm)
 - ▶ PGAS (Partitioned Global Address Space) language
 - ▶ An ancestor of UPC...
- Already on the Saclay Plateau ! ☺

Quite simple business model

- Customers need just to rewrite all their code in HyperC ☺
- Difficult entry cost... ☹
- Niche market... ☹
- American subsidiary with dataparallel datamining application acquired by Yahoo! in 1998
- Closed technology → lost for customers and... founders ☹



Present motivations: reinterpreting Moore's law (I)

The good news ☺

- Number of transistors still increasing
- Memory storage increasing (DRAM, FLASH...)
- Hard disk storage increasing
- Processors (with captors) everywhere
- Network is increasing

• The bad news ☹

- ▶ Transistors are so small they leak... Static consumption
- ▶ Superscalar and cache are less efficient compared to transistor budget
- ▶ Storing and moving information is expensive, computing is cheap: change in algorithms...
- ▶ Light's speed has not improved for a while... Hard to reduce latency

■ Chips are too big to be globally synchronous at multi GHz ☹



Present motivations: reinterpreting Moore's law (II)

- ▶ pJ and physics become very fashionable
- ▶ Power efficiency in $\mathcal{O}(\frac{1}{f})$
 - Transistors cannot be used at full speed without melting ☹️ 🏠
- ▶ I/O and pin counts
 - Huge time and energy cost to move information outside the chip ☹️

Parallelism is the only way to go...

Research is just crossing reality!

No one size fit all...

Future will be heterogeneous: GPGPU, Cell, vector/SIMD, FPGA, PIM...

But compilers are always behind... ☹️



Outline



- 1 HPC Project
- 2 Par4All
- 3 Scilab to OpenMP, CUDA & OpenCL
- 4 Results
- 5 Conclusion
- 6 Table des matières



HPC Project emergence

→ 2006: Time to be back in parallelism!

Yet another start-up... ☺

- People that met \approx 1990 at the French Parallel Computing military lab SEH/ETCA
- Later became researchers in Computer Science, CINES director and ex-CEA/DAM, venture capital and more: ex-CEO of Thales Computer, HP marketing...
- HPC Project launched in December 2007
- \approx 30 colleagues in France (Montpellier, Meudon), Canada (Montréal with Parallel Geometry) & USA (Santa Clara, CA)



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

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



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, Wild Cruncher for Scilab//...)
- Acceleration software for the WildNode
 - ▶ CPU+GPU-accelerated libraries for C/Fortran/Scilab/Matlab/Octave/R
 - ▶ Automatic parallelization for Scilab, C, Fortran...
 - ▶ Transparent execution on the WildNode
- Par4All automatic parallelization tool
- 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...)

Outline

- 
- 1 HPC Project
 - 2 Par4All
 - 3 Scilab to OpenMP, CUDA & OpenCL
 - 4 Results
 - 5 Conclusion
 - 6 Table des matières



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 \rightsquigarrow long-term commitment in a tool that needs to survive to (too fast) technology change




Expressing parallelism ?

- Solution libraries
 - ▶ Need to fit your application
- New parallel languages
 - ▶ Rewrite your applications...
- Extend sequential language with `#pragma`
 - ▶ Nicer transition
- Hide parallelism in object oriented classes
 - ▶ Restructure your applications...
- Use magical automatic parallelizer



Automatic parallelization

- Major research failure from the past...
- Untractable in the general case ☹
- Bad sequential programs? GIGO: Garbage In-Garbage Out...
- But technology widely used locally in main compilers
- To use #pragma, // languages or classes: cleaner sequential program or algorithm first...
- ... and then automatic parallelization can often work ☺
-  Par4All = automatic parallelization + coding rules
- Often less optimal performance but better time-to-market



Basic Par4All coding rules for good parallelization

(1)

- Develop a coding rule manual to help parallelization and... sequential quality!
- Par4All parallelizes loop-nests made from Fortran DO or C99 `for` loops similar to DO-loops
- Same constraints as `for`-loop accepted in OpenMP standard
- `for ([int] init-expr; var relational-op b; incr-expr)
statement`
- Increment and bounds: integer expressions, loop-invariant
- *relational-op* only `<`, `<=`, `>=`, `>`
- Do not modify loop index inside loop body
- Do not use `assert()` or compile with `-DNDEBUG` inside a loop. Assert has potential exit effect
- No `goto` outside the loop, `break`, `continue`



Basic Par4All coding rules for good parallelization

(II)

- No `exit()`, `longjump()`, `setcontext()`...
- Data structures
 - ▶ Pointers
 - Do not use pointer arithmetics
 - ▶ Arrays
 - PIPS uses integer polyhedron lattice in analysis, so us affine reference in parallelizable code
- 1 `// Good:`
 `a[2*i-3+m][3*i-j+6*n]`
- 3 `// Bad (polynomial):`
 `a[2*i*j][m*n-i+j]`
- Do not use linearized arrays
- Do not use recursion
- Prototype of coding rules report on-line on par4all.org



p4a in a nutshell

(1)

Parallelisation

p4a matmul.f

generates an OpenMP program in matmul.p4a.f

```
1  !$omp parallel do private(I, K, X)
2  C multiply the two square matrices of ones
   DO J = 1, N
4  !$omp parallel do private(K, X)
   DO I = 1, N
6     X = 0
   !$omp parallel do reduction(+:X)
8     DO K = 1, N
       X = X+A(I,K)*B(K,J)
10    ENDDO
   !$omp end parallel do
12    C(I,J) = X
   ENDDO
14  !$omp end parallel do
   ENDDO
16  !$omp end parallel do
```



p4a in a nutshell

(II)

Parallelisation with compilation

```
p4a matmul.f -o matmul
```

generates an OpenMP program `matmul.p4a.f` that is compiled with `gcc` into `matmul`

CUDA generation with compilation

```
p4a --cuda saxpy.c -o s
```

generates a CUDA program that is compiled with `nvcc`

OpenCL generation with compilation

```
p4a --opencl saxpy.c -o s
```



Basic GPU 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
- The host waits...
- Copy-out the results from the GPU to the host
- Deallocate the storage on the GPU

Generic scheme for other heterogeneous accelerators too



Rely on PIPS

(1)

- PIPS (Interprocedural Parallelizer of Scientific Programs): Open Source project from Mines ParisTech... 23-year old! ☺
- Funded by many people (French DoD, Industry & Research Departments, University, CEA, IFP, Onera, ANR (French NSF), European projects, regional research clusters...)
- One of the project that introduced polytope model-based compilation
- ≈ 456 KLOC according to David A. Wheeler's SLOCCount
- ... but modular and sensible approach to pass through the years
 - ▶ ≈ 300 phases (parsers, analyzers, transformations, optimizers, parallelizers, code generators, pretty-printers...) that can be combined for the right purpose
 - ▶ **Abstract interpretation**



Rely on PIPS

(II)

- ▶ **Polytope lattice** (**sparse** linear algebra) used for semantics analysis, transformations, code generation... with **approximations** to deal with big programs, not only
- ▶ NewGen object description language for language-agnostic automatic generation of methods, persistence, object introspection, visitors, accessors, constructors, XML marshaling for interfacing with external tools...
- ▶ Interprocedural *à la* make engine to chain the phases as needed. Lazy construction of resources
- ▶ On-going efforts to extend the semantics analysis for C
- Around 15 programmers currently developing in PIPS (Mines ParisTech, HPC Project, IT SudParis, TÉLÉCOM Bretagne, RPI) with public `svn`, `Trac`, `git`, mailing lists, IRC, Plone, Skype... and use it for many projects
- But still...
 - ▶ Huge need of documentation (even if PIPS uses literate programming...)



Rely on PIPS

(III)

- ▶ Need of industrialization
- ▶ Need further communication to increase community size



Current PIPS usage


- Automatic parallelization (Par4All C & Fortran to OpenMP)
- Distributed memory computing with OpenMP-to-MPI translation [STEP project]
- Generic vectorization for SIMD instructions (SSE, VMX, Neon, CUDA, OpenCL...) (SAC project) [SCALOPES]
- Parallelization for embedded systems [SCALOPES, SMECY]
- Compilation for hardware accelerators (Ter@PIX, SPoC, SIMD, FPGA...) [FREIA, SCALOPES]
- High-level hardware accelerators synthesis generation for FPGA [PHRASE, CoMap]
- Reverse engineering & decompiler (reconstruction from binary to C)
- Genetic algorithm-based optimization [Luxembourg university+TB]
- Code instrumentation for performance measures
- GPU with CUDA & OpenCL [TransMedi@, FREIA, OpenGPU]

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

(1)

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++) {
      save[i][j] = 0.25*(space[i - 1][j] + space[i + 1][j]
4      + space[i][j - 1] + space[i][j + 1]);
  }
```



- After:

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



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 = i; 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+1<=n, PHI1<=PHI2}>
    for(i = 0; i <= n-1; i += 1)
3  // <h_A[PHI1][PHI2]-WEXACT-{PHI1==i, i<=PHI2, PHI2+1<=n, 0<=i}>
        for(j = i; j <= n-1; j += 1)
5  // <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

More subtle approach

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)
```



Loop normalization

- Hardware accelerators use fixed iteration space (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++) {
3          save[i][j] = 0.25*(space[i - 1][j] + space[i + 1][j]
4              + space[i][j - 1] + space[i][j + 1]);
5      }

```

After

```

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

```



From preconditions to iteration clamping

(1)



- 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 ⚠



From preconditions to iteration clamping

(II)

- So we need to generate code such as

```

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

```

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 \equiv directly translation in C of preconditions on loop indices that are GPU thread indices



Optimized reduction generation

- Reduction are common patterns that need special care to be correctly parallelized

$$s = \sum_{i=0}^N x_i$$

- Reduction detection already implemented in PIPS
- Generate `#pragma omp reduce` in Par4All
- Generate GPU atomic operations



Communication optimization

- Naive approach : load/compute/store
- Useless communications if a data on GPU is not used on host between 2 kernels... ☹
- ~→ Use static interprocedural data-flow communications
 - ▶ Fuse various GPU arrays : remove GPU (de)allocation
 - ▶ Remove redundant communications

~→ p4a --com-optimization option since version 1.2



Loop fusion

- Programs \equiv often a succession of (parallel) loops
- Can be interesting to fuse loops together
 - ▶ Important for array-oriented languages: Fortran 95, Scilab, C++ parallel class...
 - ▶ Factorize control : one loop with bigger content
 - More important for heterogeneous accelerators: reduce kernel launch time
 - May avoid memory round trip
 - May cache recycling
- Use dependence graph, regions... to figure out when to fuse
- Sensible parallel promotion of scalar code to reduce parallelism interruption still to be implemented



Par4All Accel runtime

(1)

- CUDA or OpenCL 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 by design
- 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
 - `p4a_accel-OpenCL.h` for OpenCL, written in C/CPP/C++
 - `p4a_accel-OpenMP.h` that can be compiled with an OpenMP compiler for simulation on a (multicore) CPU
- Add CUDA support for complex numbers




Par4All Accel runtime

(II)

- Can be used to simplify manual programming too (OpenCL...)
 - ▶ Manual radar electromagnetic simulation code @TB
 - ▶ One code targets CUDA/OpenCL/OpenMP
- OpenMP emulation for almost free
 - ▶ Use Valgrind to debug GPU-like and communication code! (Nice side effect of source-to-source...)
 - ▶ May even improve performance compared to native OpenMP generation because of memory layout change



Outline

- 
- 1 HPC Project
 - 2 Par4All
 - 3 Scilab to OpenMP, CUDA & OpenCL
 - 4 Results
 - 5 Conclusion
 - 6 Table des matières



Scilab language

- Interpreted scientific language widely used like Matlab
- Free software
- Roots in free version of Matlab from the 80's
- Dynamic typing (scalars, vectors, (hyper)matrices, strings...)
- Many scientific functions, graphics...
- Double precision everywhere, even for loop indices (now)
- Slow because everything decided at runtime, garbage collecting
 - ▶ Implicit loops around each vector expression
 - Huge memory bandwidth used
 - Cache thrashing
 - Redundant control flow
- Strong commitment to develop Scilab through Scilab Enterprise, backed by a big user community, INRIA...
- HPC Project WildNode appliance with Scilab parallelization
- Reuse Par4All infrastructure to parallelize the code



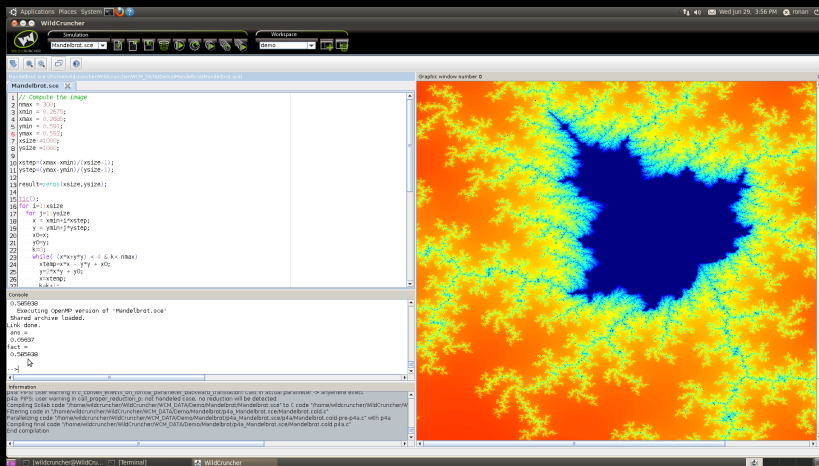
Scilab & Matlab

(1)

- Scilab/Matlab input : *sequential* or array syntax
- Compilation to C code
- Parallelization of the generated C code
- Type inference to guess (crazy ☹) semantics
 - ▶ Heuristic: first encountered type is forever
- Speedup > 1000 ☺
- Wild Cruncher: x86+GPU appliance with nice interface
 - ▶ Scilab — mathematical model & simulation
 - ▶ Par4All — automatic parallelization
 - ▶ //Geometry — polynomial-based 3D rendering & modelling
- Versions to compile to other platforms (fixed-point DSP...)



Wild Cruncher — Scilab parallelization



Outline

- 1 HPC Project
- 2 Par4All
- 3 Scilab to OpenMP, CUDA & OpenCL

- 4 Results
- 5 Conclusion
- 6 Table des matières

Stars-PM



- *Particle-Mesh* N-body cosmological simulation
- C code from Observatoire Astronomique de Strasbourg
- Use FFT 3D
- Example given in `par4all.org` distribution



Stars-PM time step

```

1 void iteration(coord pos[NP][NP][NP],
2               coord vel[NP][NP][NP],
3               float dens[NP][NP][NP],
4               int data[NP][NP][NP],
5               int histo[NP][NP][NP]) {
6     /* Split space into regular 3D grid: */
7     discretisation(pos, data);
8     /* Compute density on the grid: */
9     histogram(data, histo);
10    /* Compute attraction potential
11       in Fourier's space: */
12    potential(histo, dens);
13    /* Compute in each dimension the resulting forces and
14       integrate the acceleration to update the speeds: */
15    forcex(dens, force);
16    updatevel(vel, force, data, 0, dt);
17    forcey(dens, force);
18    updatevel(vel, force, data, 1, dt);
19    forcez(dens, force);
20    updatevel(vel, force, data, 2, dt);
21    /* Move the particles: */
22    updatepos(pos, vel);
23 }

```



Stars-PM & Jacobi results

- 2 Xeon Nehalem X5670 (12 cores @ 2,93 GHz)
- 1 GPU nVidia Tesla C2050
- Automatic call to CuFFT instead of FFTW (stubs...)
- 150 iterations of Stars-PM

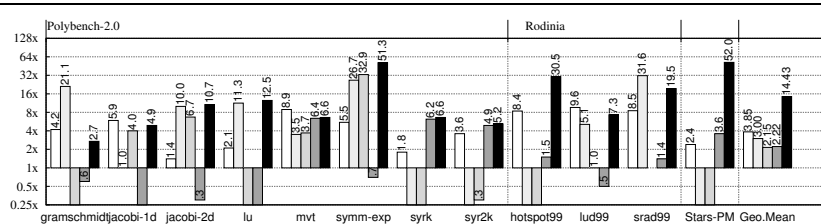
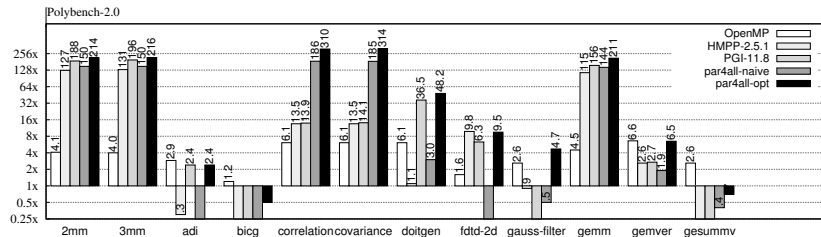
Speed-up	p4a	Simulation Cosmo.			Jacobi
		32 ³	64 ³	128 ³	
Sequential (time in s)	(gcc -O3)	0.68	6.30	98.4	24.5
OpenMP 6 threads	--openmp	4.25	4.92	5.9	1.78
CUDA base	--cuda	0.77	1.21	3.13	0.36
Optim. comm. 1.1	--cuda	3.4	5.38	11	3.8
	--com-opt.				
Reduction Optim. 1.1.2	--cuda	6.8	19.7	46.9	6.4
	--com-opt.				
Manual optim.	(gcc -O3)	13.6	24.2	54.7	

p4a 1.1.2 introduce generation of CUDA atomic updates for PIPS detected reductions. Other solution to investigate: CuDPP call generation



Benchmark results

With Par4All 1.2, CUDA 4.0, WildNode 2 Xeon Nehalem X5670 (12 cores @ 2.93 GHz) with nVidia C2050



From par4all.org distribution, in examples/Benchmarks

Outline

- 
- 1 HPC Project
 - 2 Par4All
 - 3 Scilab to OpenMP, CUDA & OpenCL
 - 4 Results
 - 5 Conclusion
 - 6 Table des matières



Saint Cloud gatekeeper & massive virtual I/O



Some events in the area

- Creation of « HPC & GPU Supercomputing Group of Paris Meetup »
 - ▶ 25/01/2012 @ Mines ParisTech, jardins du Luxembourg, Paris
 - ▶ + Par4All presentation
 - ▶ meetup.com/HPC-GPU-Supercomputing-Group-of-Paris-Meetup
- Wild Cruncher UV
 - ▶ Scilab parallelization on SGI UV
 - ▶ 14/02/2012, SGI breakfast @ Novell France, Tour Franklin, La Défense



Conclusion

(1)

- GPU (and other heterogeneous accelerators): impressive peak performances and memory bandwidth, power efficient
- Domain is maturing: any languages, libraries, applications, tools... Just choose the good one ☺
- Real codes are often not well written to be parallelized... even by human being ☹
- At least writing clean C99/Fortran/Scilab... code should be a prerequisite
- Take a positive attitude... Parallelization is a good opportunity for deep cleaning (refactoring, modernization...) → improve also the original code
- Open standards to avoid sticking to some architectures
- Need software tools and environments that will last through business plans or companies





- Open implementations are a warranty for long time support for a technology (cf. current tendency in military and national security projects)
- p4a motto: keep things simple
- Open Source for community network effect
- Easy way to begin with parallel programming
- Source-to-source
 - ▶ Give some programming examples
 - ▶ Good start that can be reworked upon
- ⚠ Entry cost
- ⚠ ⚠ ⚠ Exit cost! ☹
 - ▶ Do not loose control on *your* code and *your* data !
- HPC Project is hiring ☺



Outline

- 
- 1 HPC Project
 - 2 Par4All
 - 3 Scilab to OpenMP, CUDA & OpenCL
 - 4 Results
 - 5 Conclusion
 - 6 Table des matières





SOLOMON

POMP & PompC @ LI/ENS 1987–1992

HyperParallel Technologies (1992–1998)

HyperParallel Technologies (1992–1998)

Present motivations: reinterpreting Moore's law

1 HPC Project

Outline

HPC Project emergence

HPC Project hardware: WildNode from Wild Systems

HPC Project software and services

2 Par4All

Outline

We need software tools

Expressing parallelism ?

Automatic parallelization

Basic Par4All coding rules for good parallelization

p4a in a nutshell

Basic GPU execution model

Rely on PIPS

Current PIPS usage

Automatic parallelization

Outlining

From array regions to GPU memory allocation

Communication generation

Loop normalization

2	From preconditions to iteration clamping	32
3	Optimized reduction generation	34
4	Communication optimization	35
5	Loop fusion	36
6	Par4All Accel runtime	37

3 Scilab to OpenMP, CUDA & OpenCL

Outline

8	Scilab language	39
---	-----------------	----

9	Scilab & Matlab	40
---	-----------------	----

10	Wild Cruncher — Scilab parallelization	41
----	--	----

11		42
----	--	----

4 Results

Outline

12	Stars-PM	43
----	----------	----

13	Stars-PM time step	44
----	--------------------	----

14	Stars-PM & Jacobi results	45
----	---------------------------	----

15	Benchmark results	46
----	-------------------	----

16		47
----	--	----

5 Conclusion

Outline

18		48
----	--	----

20	Saint Cloud gatekeeper & massive virtual I/O	49
----	--	----

21	Some events in the area	50
----	-------------------------	----

24	Conclusion	51
----	------------	----

25		
----	--	--

6 Table des matières

Outline

26		53
----	--	----

30		54
----	--	----

31	Vous êtes ici !	
----	------------------------	--

