Programming environments for big data processing on modern parallel architectures

Ronan KERYELL
HPC Project

2011/06/09

Le déluge de données, comment en tirer parti ?
Séminaire Aristote

- 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 😞

- 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

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 as not improved for a while... Hard to reduce latency
  - Chips are too big to be globally synchronous at multi GHz 😊
- pJ and physics become very fashionable
Present motivations: reinterpreting Moore’s law (II)

- Power efficiency in $O\left(\frac{1}{T}\right)$
  - Transistors cannot be used at full speed without melting 😞
  - Heat
- I/O and pin counts
  - Huge time and energy cost to move information outside the chip 😞

- Rotating hard disk with 1D density $d$ increase
  - Storage in $O(d^2)$
  - But track speed only $O(d)$
  - Reading all the disk in $O\left(\frac{1}{d}\right)$ 😐
Parallelism is the only way to go...

- Research is just crossing reality!
- Scaring... 😞
- Exciting! 😊

No one size fit all...

Future will be heterogeneous
Edsger Dijkstra, 1972 Turing Award Lecture, « The Humble Programmer »

“To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem.”


⚠️ But... it was before parallelism democratization! 😊
Time to be back in parallelism!

- Good time for more start-ups! 😊
- ≈ 2006: thinking to yet another start-up...
- People that met ≈ 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
- ≈ 30 colleagues in France (Montpellier, Meudon), Canada (Montréal with Parallel Geometry) & USA (Mountain View)
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

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, WildCruncher for Scilab/...)
- Acceleration software for the WildNode
  - GPU-accelerated libraries for C/Fortran/Scilab/Matlab/Octave/R
  - Transparent execution 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...)
Efficient big data architectures

- Massive parallelism
- Use right processing elements for the right tasks (efficiency)
- Avoid moving data (expensive, slow)
- Avoid storing data: on-the-fly processing of data to be correlated
- Use memory hierarchies
- Distributed computing with smart routers
Outline

1. Hardware architectures
   - Classical multicores
   - GPU
   - MP-SoC underworlds

2. Software environments
   - Programming challenges
   - Multicores
   - GPU
   - Application libraries

3. Par4All
   - GPU code generation
   - Code generation for SCMP

4. Conclusion
Outline

1. Hardware architectures
   - Classical multicores
   - GPU
   - MP-SoC underworlds

2. Software environments
   - Programming challenges
   - Multicores
   - GPU
   - Application libraries

3. Par4All
   - GPU code generation
   - Code generation for SCMP

4. Conclusion
Intel Nehalem

- x86-64 new microarchitecture
- With on-chip memory controllers like AMD Opteron: \( \uparrow \) bandwidth, \( \downarrow \) latency
- SSE4.2 instruction set: strings (XML parser...), comparisons (data-mining), CRC & cryptography (protocols)
- Power consumption fine tuning with many sensors (power, temperature). Possible to speed up when less cores are used (turbo)
- Xeon X7560 (Beckton microarchitecture), 2010
  - 8 cores @ 2.27 GHz (+ turbo 2.666 GHz) + SMT 2 threads (HyperThreading) with 256 KB L2 cache/core, 32D+32I KB cache/core
  - 24 MB L3 cache
  - 4 Quick Path Interconnects (QPI) @ 6.4 GT/s to play Lego with processors & accelerators \( \approx \) HyperTransport d’AMD
Hardware architectures

Classical multicores

Intel Nehalem

- 4 DDR3-1333 MHz memory controllers
- 130 W
- $3692 March 30, 2011
- 2.3 Gtr 45 nm 684 mm^2
Hardware architectures

Intel Nehalem

• Classical multicores

- Core 1
- Core 2
- Core 3
- Core 4
- Core 5
- Core 6
- Core 7
- Core 8

Cbox
(LLC Coherence Engine)

24M Last Level Cache (L3 Cache)

Dual Intel® SMI Channels

Sbox
(Caching Agent 1)

Sbox
(Caching Agent 2)

Pbox
(Physical Layer)

Mbox
(Memory Controller)

Bbox
(Home Agent 1)

Rbox
(Router)

4 Intel® QPI links

Dual Intel® SMI Channels

Pbox
(Physical Layer)

Mbox
(Memory Controller)

Bbox
(Home Agent 2)

Ubox
(System Config Controller)
Up to 8 sockets: Node Controllers optional

Larger than 8 sockets: Node Controllers required

= Intel® Xeon® processor 7500 series
Hardware architectures

AMD Opteron 6180 (2011)

- x86-compatible instruction set
- 64-bit mode that double also register numbers
- Out-of-order superscalar execution with 9 instructions/cycle
  - 3 integer instructions
  - 3 address generators
  - 3 floating computation operators (+, *, memory)

- Opteron 6180 SE, 02/2011, $1514
  - 12 cores @ 2.5 GHz, 512 KB/core L2 cache
  - 2 × 6 MB L3 cache
  - 21 GB/s DDR3 memory
  - 4 x 6.4 GT/s HyperTransport
  - 1.8 Gtr 45 nm 692 mm²
  - 140 W
IBM Power 7 (2010)

- 4 chips per quad-chip module
- 8 cores per chip @ 4.0 GHz (4.25 GHz in TurboCore mode with 4 cores)
- 1.2 Gtr 45 nm SOI process, 567 mm²
- 4 SMT threads per core
- 32I+32D kB L1 cache/core
- 256 kB L2 cache/core
- 32 MB L3 cache in eDRAM
- 100 GB/s DDR3 memory
- 12 execution units per core:
  - 2 fixed-point units
  - 2 load/store units
  - 4 double-precision floating-point units
  - 1 vector unit supporting VSX
  - 1 decimal floating-point unit
  - 1 branch unit
  - 1 condition register unit
Outline

1. Hardware architectures
   - Classical multicores
   - GPU
   - MP-SoC underworlds

2. Software environments
   - Programming challenges
   - Multicores
   - GPU
   - Application libraries

3. Par4All
   - GPU code generation
   - Code generation for SCMP

4. Conclusion

Programming environments for big data processing on modern parallel architectures
Off-the-shelf AMD/ATI Radeon HD 6970 GPU

- 2.64 billion 40nm transistors
- 1536 stream processors @ 880 MHz, 2.7 TFLOPS SP, 675 GFLOPS DP
- + External 1 GB GDDR5 memory 5.5 Gt/s, 176 GB/s, 384b GDDR5
- 250 W on board (20 idle), PCI Express 2.1 x16 bus interface
- OpenGL, OpenCL
- ∃ Radeon HD 6990 double chip card

More integration:

- Llano APU (FUSION Accelerated Processing Unit) : x86 multicore + GPU 32nm, OpenCL
Radeon HD 6870 — big picture
Off-the-shelf nVidia Tesla Fermi M2090 & GTX580

- GF110: 3 billion 40nm tr.
- 512 thread processors @ 1300 MHz, 1.3 TFLOPS SP, 666 GFLOPS DP
- + External 6 GB GDDR5 ECC memory 3.7 Gt/s, 177 GB/s. Less if using ECC

- 247 W on board PCI Express 2.1 x16 bus interface
- OpenGL, OpenCL, CUDA

Programming environments for big data processing on modern parallel architectures

HPC Project

Ronan KERRYELL
GF100 Stream Multiprocessor

Fermi GF100 Streaming Multiprocessor (SM)

SM (Streaming Multiprocessor)

PolyMorph Engine
- Vertex Fetch
- Tessellator
- Viewport Transform
- Attribute Setup
- Stream Output

Instruction Cache

2-Issue Instruction Unit
- Warp Scheduler
- Warp Dispatcher
- Warp Scheduler
- Warp Dispatcher

CUDA cores x16
(Single/Double Precision)

CUDA cores x16
(Single/Double Precision)

SFUs x4
(Special Function Unit)

Load/Store Units x16

Texture Address Generators x4

Texture Filtering Units x4

Texture Cache 12KB

Register Files 128KB
(32-bit Register x 32768)

64KB Configurable L1 Cache/Shared Memory
(16KB Cache + 48KB SM/
48KB Cache + 16KB SM)
Outline

1. Hardware architectures
   • Classical multicores
   • GPU
   • MP-SoC underworlds

2. Software environments
   • Programming challenges
   • Multicores
   • GPU
   • Application libraries

3. Par4All
   • GPU code generation
   • Code generation for SCMP

4. Conclusion
ARM yourself

- Do some computations where the captors are...
- Smartphone and other sensor networks
- Trade-off between communication energy and inside/remote computations
- Texas Instrument OMAP4470 announced on 2011/06/02
  - 2 ARM Cortex-A9 MPCores @ 1.8GHz with Neon vector instructions
  - 2 ARM Cortex-M3 cores (low-power and real-time responsiveness, multimedia, avoiding to wake up the Cortex-A9...)
  - **SGX544 graphics core with OpenCL 1.1 support**, with 4 USSE2 core @ 384 MHz producing each 4 FMAD/cycle: 12.3 GFLOPS
  - 2D graphics accelerator
  - 3 HD displays and up to QXGA (2048x1536) resolution + stereoscopic 3D
  - Dual-channel, 466 MHz LPDDR2 memory
• Current course to have non-x86 servers based on ARM...
• Experiments on low power clusters
• Think to evaluate power consumption on your application
Tilera TilePro64

Programming environments for big data processing on modern parallel architectures
Tilera TilePro64

- Processing power with good network interfaces
  - Interactive data analysis
  - Video/audio codec
  - DPI, IDS, IPsec...
- 8x8 processors with SMP, partitionable
- 32-bit VLIW with SIMD mode
- 700-866 MHz: 443 GOPS 8 bits, 23 W
- 64 DDR2 controllers 25.6 GB/s
- $2 \times 10$ GbE XAUI + IP session hash distribution
- SMP Linux or bare bone per tile
- C/C++ gcc compiler
- TMC library for hardcore support (2D network...)
- Eclipse support with graphical simulator
- OpenMP
Hardware architectures

MPPA de Kalray (the French touch!)

- 256 VLIW processors per chip, 16 clusters of 16 processors
- Shared memory and NoC with DMA
- 28 nm CMOS technology, \( \approx 5 \text{ W} @ 400 \text{ MHz} \)
- FPU 32/64 bits IEEE 754: 205 GFLOPS SP, 1 TOPS 16 bits
- 2\( \times \) 64-bit DDR3 memory controllers for high bandwidth main memory transfers
- 2\( \times \) 40 Gb/s or 8\( \times \) 10 Gb/s Ethernet controller
MPPA de Kalray (the French touch!)

- 2× 8-lane PCI Express Gen 3
- 4× 4–8-lane Interlaken interfaces for multi-MPPA chip system integration (8 MPPA/PCIe board) or connection to external FPGAs, I/O...
- Linux or bare metal with AccessCore library
- Multi-core compiler (gcc 4.5), simulator, debugger (gdb), profiler
- Eclipse IDE
- Programming from high-level C-based language
- AccessCore library
Field-programmable gate array with bitstream configuration in memory

- Xilinx Virtex 7
  - 2M logic cells (6-LUT), 28 nm
  - 85Mb block RAM
  - 5280 DSP slices (6.7 TFMA/s)
  - 96 transceiver @ 28Gb/s: 2.8 Tb/s
  - PCIe gen3 ×8
  - 1200 pins

- Radar, communications, HPC, datamining, bioinformatics...
- VHDL-to-bitstream compiler... but hard work
- Dynamic partial reconfiguration
- ∃ C-to-VHDL compilers
  - Riverside Optimizing Compiler for Configurable Computing (ROCCC): open source
  - Impulse C
  - Catapult C
  - Cynthesizer
  - ...
Convey HC-1<sup>ex</sup>

- Intel Xeon quad-core @2.13 GHz with 128 GB
- 4 Xilinx Virtex 6 LX760 FPGA with 128 GB DDR2
- 1520 W in 3U rack
- Linux
- Various instruction sets ("personality")
- Fortran/C/C++ vectorizing compiler replacing complex instructions by FPGA vectorized implementation
- Possible to design its own instruction set (ROCCC...)
- 401× speed-up on Smith-Waterman algorithm compared to x86
Anton computer from D.E. Shaw

1. Create a hedge fund
2. Earn a lot of money
3. Spend it by creating a 500+ people start-up in bioinformatics
4. Build from scratch (ASIC) a computer for solving special issues
PacketShader

- Use Linux PC + GPU as a router with processing power
  - 2 4-core Nehalem @2.66 GHz + 4 10 GigE NIC + 2 GPU GTX480
- 40 Gb/s routing even with 64-byte packets
- 8–20 Gb/s IPsec tunneling
- Linux IP stack to slow; own raw device driver for Intel 82598/82599 NIC
  - Extract Ethernet flow into user mode
  - Huge recycled packet buffers
  - Batch processing to amortize overhead
  - NUMA-aware processing: packets processed by NIC-local CPU in its RAM, aligned in cache

- Current bottleneck: IOH chipset

http://shader.kaist.edu/packetshader
OpenFlow

- Need for advanced routing with computing capabilities
- PC or Tilera-like with few 10+ Gb/s Ethernet or Infiniband: bandwidth maybe OK but not enough links
- Open standard to interact with existing routers & switch: OpenFlow [http://www.openflow.org](http://www.openflow.org)
- Possible to extract *minimal* flows to feed PC/GPU/MP-SoC/FPGA accelerators and reinject results in the router
- OpenFlow implemented by many router companies
- Possible with non-OpenFlow but less flexible and non portable
Outline

1. Hardware architectures
   - Classical multicores
   - GPU
   - MP-SoC underworlds

2. Software environments
   - Programming challenges
   - Multicores
   - GPU
   - Application libraries

3. Par4All
   - GPU code generation
   - Code generation for SCMP

4. Conclusion
Outline

1. Hardware architectures
   - Classical multicores
   - GPU
   - MP-SoC underworlds

2. Software environments
   - Programming challenges
   - Multicores
   - GPU
   - Application libraries

3. Par4All
   - GPU code generation
   - Code generation for SCMP

4. Conclusion
Parallel application dwarfs

- [http://view.eecs.berkeley.edu/](http://view.eecs.berkeley.edu/) « The Landscape of Parallel Computing Research: A View From Berkeley »
- Try to capture typical examples to analyze and design new architectures & applications

<table>
<thead>
<tr>
<th>Dwarf</th>
<th>Performance Limit: Memory Bandwidth, Memory Latency, or Computation?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Dense Matrix</td>
<td>Computationally limited</td>
</tr>
<tr>
<td>2 Sparse Matrix</td>
<td>Currently 50% computation, 50% memory BW</td>
</tr>
<tr>
<td>3 Spectral (FFT)</td>
<td>Memory latency limited</td>
</tr>
<tr>
<td>4 N-Body</td>
<td>Computationally limited</td>
</tr>
<tr>
<td>5 Structured Grid</td>
<td>Currently more memory bandwidth limited</td>
</tr>
<tr>
<td>6 Unstructured Grid</td>
<td>Memory latency limited</td>
</tr>
<tr>
<td>7 MapReduce</td>
<td>Problem dependent</td>
</tr>
<tr>
<td>8 Combinational Logic</td>
<td>CRC problems BW; crypto problems computationally limited</td>
</tr>
<tr>
<td>9 Graph traversal</td>
<td>Memory latency limited</td>
</tr>
<tr>
<td>10 Dynamic Programming</td>
<td>Memory latency limited</td>
</tr>
<tr>
<td>11 Backtrack and Branch+Bound</td>
<td>?</td>
</tr>
<tr>
<td>12 Construct Graphical Models</td>
<td>?</td>
</tr>
<tr>
<td>13 Finite State Machine</td>
<td>Nothing helps!</td>
</tr>
</tbody>
</table>
Extracting parallelism in applications...

- The implicit attitude
  - Hardware: massively superscalars processors
  - Software: auto-parallelizing compilers

- The (±) explicit attitude
  - Languages (± extensions): OpenMP, UPC, HPF, Co-array Fortran (F- -), Fortran 2008, X10, Chapel, Fortress, Matlab, SciLab, Octave, Mapple, LabView, nVidia CUDA, AMD/ATI Stream (Brook+, Cal), OpenCL, HMPP, *insert your own preferred language here* ...
  - Framework: MapReduce, Hadoop...
  - Libraries: application-oriented (mathematics, coupling...), parallelism (MPI, concurrency *pthreads*, SPE/MFC on Cell...), Multicore Association MCAPI, objects (parallel STL, TBB, Ct...)
... but multidimensional heterogeneity!

Welcome into Parallel Hard-Core Real Life 2.0!

- Heterogeneous execution models
  - Multicore SMP ± coupled by caches
  - SIMD instructions in processors (Neon, VMX, SSE4.2, 3DNow!, LRBni...)
  - Hardware accelerators (MIMD, MISD, SIMD, SIMT, FPGA...)

- New heterogeneous memory hierarchies
  - Classic caches/physical memory/disks
  - Flash SSD is a new-comer to play with
  - NUMA (*Non Uniform Memory Access*) : sockets-attached memory banks, remote nodes...
  - Peripherals attached to sockets : NUPA (*Non Uniform Peripheral Access*). GPU on PCIe ×16 in this case...
  - If non-shared memory: remote memory, remote disks...
  - Inside GPU : registers, local memory, shared memory, constant memory, texture cache, processor grouping, locked physical pages, host memory access...

- Heterogeneous communications
  - Anisotropic networks
  - Various protocols

Several dimensions to cope with at the same time 😊
Outline

1. Hardware architectures
   - Classical multicores
   - GPU
   - MP-SoC underworlds

2. Software environments
   - Programming challenges
   - Multicores
     - GPU
     - Application libraries

3. Par4All
   - GPU code generation
   - Code generation for SCMP

4. Conclusion
OpenMP

- « Le multithread pour les nuls » 😊
- Vise machines à mémoire partagée
- Sauf si programmation système compliquée, pas besoin de faire de la programmation de threads explicites
- Idée : saupoudrer un programme de directives pour aider compilateur à paralléliser
- Philosophie : #pragma pragmatisme avec une certaine élégance esthétique
- 🚧 Si pas de directives, pas de parallélisme exploité (a priori)
- 🚧 🚧 🚧 Directive ≡ déclaration sur l’honneur
- Langages supportés : Fortran et C/C++

http://openmp.org
Modèle d’exécution d’OpenMP

http://openmp.org/wp/openmp-specifications

- Exécution parallèle SPMD basée sur le *fork/join*

- Création de thread implicite ou explicite avec des *directives*
Astuce : un programme OpenMP peut être exécuté comme
• Programme séquentiel
• Programme parallèle
→ portabilité, coût de sortie nul ! 😊

• Threads créées dans des sections parallel et stoppées à la fin avec une barrière
• Constructions de synchronisation et dans bibliothèque
• Contrôle de l’environnement d’exécution par variables d’environnement et fonctions de bibliothèque
Exemple

```c
#pragma omp parallel default(none) \ 
    shared(n,x,y) private(i)
{
    /* Ceci s'exécute sur plusieurs threads en */
    #pragma omp for
    for (i=0; i<n; i++)
        // Les itérations sont réparties sur les threads
        x[i] += y[i];
    /* Synchronisation implicite ici */
}/*--- End of parallel region ---*/
/* Synchronisation implicite ici */
```
Task en OpenMP 3.0

- Rajout de la notion de tâches explicites ≡ bout de programme exécuté sur une thread
- Tâche créée dans thread par construction task (TBB, Cilk...)

```c
#pragma omp parallel
{
    #pragma omp single private(p)
    {
        p = listhead;
        while (p) {
            #pragma omp task
            {
                process (p);
            }
            p = next(p);
        }
    }
}
```

- Extensions en vue (target...)

Software environments

Multicores
Multimedia SIMD extensions

- Multimedia and telecommunication market
  - GIF image: 8-bit pixels
  - RGB\(\alpha\) true-color image: 4 \(\times\) 8 bit pixels
  - Phone A- or \(\mu\)-law: 8-bit samples
  - CD-quality sound: 2 \(\times\) 16-bit samples

- In general purpose processors, under-used transistors on these applications (double precision multiplier...)

  Idea: 128-bit data viewed as independent vectors of 16 independent 8-bit elements or 8 16-bit or... 2 DP-float or 4 SP-float

- Add SIMD to general purpose processors (i860, SSE 4.2 Core i7, AMD 3Dnow!, VMX Power, ARM, SPARC...) and Cell for numerical computing, strings, data compaction, cryptography...

- SSE4.2 at 3.2 GHz: 2 128-bit operations/cycle \(\sim\) 819 GOP1b/s, 102 GOP8b/s *per core!*
Multimedia SIMD extensions

- Programming
  - Quite complex 😞 because many *ad hoc* instructions, saturation arithmetic...
  - Just do it... in assembly language 😊
  - Intrinsics functions in C/C++ (GCC, Intel...)
  - C/C++ extensions with new vector data types (GCC, Intel...)
  - Auto-vectorizing (IBM xlc, Intel, GCC, generic tools such as PIPS)
  - Use already optimized libraries

- Available on all machines 😊

Software environments

• Multicores

• Multimedia SIMD extensions

(II)
• Le passage de message pour les nuls ! 😊
• Bibliothèque de fonctions de communication disponible pour de nombreux langages et systèmes d’exploitation
• Portabilité et nivellement par le bas : programmation de SMP aussi en MPI...
• Ressources
  ► http://en.wikipedia.org/wiki/Message_Passing_Interface
  ► MPI Forum http://www.mpi-forum.org
  ► MPICH : A Portable Implementation of MPI
    http://www-unix.mcs.anl.gov/mpi/mpich/
  ► LAM / MPI Parallel Computing http://www.mpi.nd.edu/lam/
  ► MPE Graphics–Scalable X11 Graphics in MPI
    http://www-fp.mcs.anl.gov/~lusk/papers/mpe/
  ► Livre « Using MPI: Portable Parallel Programming with the Message-Passing Interface »
    http://www-unix.mcs.anl.gov/mpi/usingmpi/
Faute de temps je n’utilise plus mes transparents trop complets
http://enstb.org/~keryell/cours/MR2/IAHP/MPI

J’utilise « An Introduction to MPI Parallel Programming with the Message Passing Interface » de William Gropp & Ewing Lusk
Thread Building Blocks (TBB)

http://en.wikipedia.org/wiki/TBB from Intel

- Template library (à la STL)
- Open and commercial Versions
- Algorithms (for, reduce, pipeline, scan...), containers, memory allocators, mutual exclusion, atomic operations, schedulers, profiling...
- Work stealing between tasks
- Orthogonal to OpenMP & MPI
Thread Building Blocks (TBB)

```cpp
class ApplyFoo {
    float* const my_a;

public:
    ApplyFoo(float* a) : my_a(a) {}
    void operator() (const tbb::blocked_range<size_t>& r) const {
        for (size_t i=r.begin(); i != r.end(); ++i)
            Foo(my_a[i]);
    }
}

void ParallelApplyFoo(float a[], size_t n) {
    tbb::parallel_for (tbb::blocked_range<size_t>(0,n),
        ApplyFoo(a),
        tbb::auto_partitioner());
}

• Need a deep code restructuring if an application is not in a STL spirit
```
Basic GPU programming 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
GPU execution model

Grid

Block (0, 0)  Block (1, 0)  Block (2, 0)

Block (0, 1)  Block (1, 1)  Block (2, 1)

Block (1, 1)

Thread (0, 0)  Thread (1, 0)  Thread (2, 0)  Thread (3, 0)

Thread (0, 1)  Thread (1, 1)  Thread (2, 1)  Thread (3, 1)

Thread (0, 2)  Thread (1, 2)  Thread (2, 2)  Thread (3, 2)

Multithreaded CUDA Program

GPU with 2 Cores

Core 0  Core 1

GPU with 4 Cores

Core 0  Core 1  Core 2  Core 3

Block 0  Block 1  Block 2  Block 3

Block 4  Block 5  Block 6  Block 7
From hardware constraints to programming style

- GPU computes fast but connected to CPU with slow PCI link
  - Avoid exchanging too much data between CPU & GPU or compute on the CPU
  - Possible to overlap communications with computations (more complex programming)
- Many SIMD engines (multiprocessors) at least as much blocks of threads
- Memory hierarchy is quite complex and... visible!
  - Use (quite limited) local registers by recycling local data
  - Memory is accessed in huge lines program to use all the elements of the line
  - If not possible, try to reorganize data in the shared memory around read/write (matrix transposition...)
  - Recently added caches help too
From hardware constraints to programming style

- Memory is far far away (800+ cycles) $\implies$ use a lot of threads per block (but limited resources reduce block numbers) to overlap memory access with other computations.

- Computing is fast, memory is slow. Rethink algorithms...

- SIMD machine, only one control flow $\implies$ predicated

```python
1  if (cond[i])
    b[i] = a[i];
3  else
    b[i] = -a[i] + 1;
```

- Some hardware optimizations if in a SIMD warp there is no execution $\implies$ if possible sort `false/true` elements.
Programmation CUDA

- Data-parallel extension to a C++ subset
- Target nVidia GPU and x86 multicores
- 2-level parallelism: threads in blocks of threads + block-tiling
- In a block of threads: communication through shared memory and synchronization via \_\_syncthreads() 
- Complex heterogeneous memory layout (GPU...)

```c
__global__ void add_matrix_gpu(float *a, float *b, float *c, int N) {
    int i=blockIdx.x*blockDim.x+threadIdx.x;
    int j=blockIdx.y*blockDim.y+threadIdx.y;
    int index =i+j*N;

    if( i < N && j < N)
        c[index]=a[index]+b[index];
}

void main() {
    float ha[N][N], hb[N][N], hc[N][N];
```
/* Allocate array on the GPU with cudaMalloc */
float *a, *b, *c;
cudaMalloc((void **) &a, sizeof(float)*N*N);
cudaMalloc((void **) &b, sizeof(float)*N*N);
cudaMalloc((void **) &c, sizeof(float)*N*N);

cudaMemcpy(a, ha, sizeof(float)*N*N, cudaMemcpyHostToDevice);
cudaMemcpy(b, hb, sizeof(float)*N*N, cudaMemcpyHostToDevice);

// Describe iteration tiling (2D strip-mining)
dim3 dimBlock (blocksize,blocksize);
dim3 dimGrid (N/dimBlock.x,N/dimBlock.y);
add_matrix_gpu<<<dimGrid,dimBlock>>>(a,b,c,N);
cudaMemcpy(c, hc, sizeof(float)*N*N, cudaMemcpyDeviceToHost);
}

- Need some heavy code restructuring
- ∃ other version: CUDA driver, similar to OpenCL
OpenCL

- Language based on a C$_{99}$ subset
- Started by Apple to *unify* parallel use (multicores, GPGPU...)
  ~ similar to OpenGL & OpenAL
- Followed by AMD/ATI and nVidia
- Data-parallelism and control-parallelism (1–3-dimensions) according to targets
- Kernel oriented computations on streams
- Complex split memory model (GPGPU...) but CPU compliant too
- New types (vectors, images...)

Software environments

HPC Project

Ronan Keryell

61 / 101
This kernel computes FFT of length 1024.
The 1024 length FFT is decomposed into calls to a radix 16 function, another radix 16 function and then a radix 4 function */

__kernel void fft1D_1024 (__global float2 *in, __global float2 *out, __local float *sMemx, __local float *sMemy) {
  int tid = get_local_id(0);
  int blockIdx = get_group_id(0) * 1024 + tid;
  float2 data[16];
  // starting index of data to/from global memory
  in = in + blockIdx; out = out + blockIdx;
  globalLoads(data, in, 64); // coalesced global reads
  fftRadix16Pass(data);       // in-place radix-16 pass
  twiddleFactorMul(data, tid, 1024, 0);
  // local shuffle using local memory
  localShuffle(data, sMemx, sMemy, tid,
               (((tid & 15) * 65) + (tid >> 4)));
  fftRadix16Pass(data);       // in-place radix-16 pass
  twiddleFactorMul(data, tid, 64, 4); // twiddle factor multiplication
  localShuffle(data, sMemx, sMemy, tid,
               (((tid >> 4) * 64) + (tid & 15)));
  // four radix-4 function calls
  fftRadix4Pass(data); fftRadix4Pass(data + 4);
  fftRadix4Pass(data + 8); fftRadix4Pass(data + 12);
// coalesced global writes
globalStores(data, out, 64);
}

context = clCreateContextFromType(CL_DEVICE_TYPE_GPU);

queue = clCreateWorkQueue(context, NULL, NULL, 0);

memobjs[0] = clCreateBuffer(context, CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
                           sizeof(float)*2*num_entries, srcA);
memobjs[1] = clCreateBuffer(context, CL_MEM_READ_WRITE,
                           sizeof(float)*2*num_entries, NULL);

program = clCreateProgramFromSource(context, 1,
                                     &fft1D_1024_kernel_src, NULL);

clBuildProgramExecutable(program, false, NULL, NULL);

kernel = clCreateKernel(program, "fft1D_1024");

global_work_size[0] = n;
local_work_size[0] = 64;
range = clCreateNDRangeContainer(context, 0, 1, 
global_work_size, local_work_size);

// set the args values
clSetKernelArg(kernel, 0, (void *)&memobjs[0], sizeof(cl_mem), NULL);
clSetKernelArg(kernel, 1, (void *)&memobjs[1], sizeof(cl_mem), NULL);
clSetKernelArg(kernel, 2, NULL,
sizeof(float)*(local_work_size[0]+1)*16, NULL);
clSetKernelArg(kernel, 3, NULL,
sizeof(float)*(local_work_size[0]+1)*16, NULL);

// execute kernel
clExecuteKernel(queue, kernel, NULL, range, NULL, 0, NULL);

- Need a lot of code restructuring
### CUDA
- Appeared first
- Language basis not well defined: C++ like, rather C89 and not C99
- Painful to translate C99 to C89 and keeping clean sources
- Rather limited to 2D threads
- nVidia GPU only

### OpenCL
- Standard backed by many companies
- C99 based clean
- 3D threads with less constraints
- More verbose API (kernel call...)
- Kernel source code outside of host source: more complex
- Fast spreading in embedded computing world (MP-SoC)
Outline

1. Hardware architectures
   - Classical multicores
   - GPU
   - MP-SoC underworlds

2. Software environments
   - Programming challenges
   - Multicores
   - GPU
   - Application libraries

3. Par4All
   - GPU code generation
   - Code generation for SCMP

4. Conclusion
Bibliothèques mathématiques

- Existent pour différentes architectures !
- Souvent bibliothèques constructeurs optimisées pour leurs machines
  - Intel Math Kernel Library (MKL)
  - AMD Performance Library (Framewave libre)
- FFT : FFTW (*Fastest Fourier Transform in the West*, en C généré par du OCaml)...
- Beaucoup d’algèbre linéaire
  - BLAS (*Basic Linear Algebra Subprograms*) et PBLAS
  - LAPACK (*Linear Algebra PACKage*)
  - ScaLAPACK : version SPMD avec MPI
  - SuperLU : solution directe de gros systèmes creux
  - PETSc (*Portable, Extensible Toolkit for Scientific Computation*) : large spectre, au dessus de MPI
Use the Source, Luke...

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

<table>
<thead>
<tr>
<th>What has survived for 50+ years?</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Fortran programs...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>What has survived for 40+ years?</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>IDL, Matlab, Scilab...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>What has survived for 30+ years?</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>C programs, Unix...</td>
<td></td>
</tr>
</tbody>
</table>

- A lot of legacy code could be pushed onto parallel hardware (accelerators) with automatic tools...
- Need automatic tools for source-to-source transformation to leverage existing software tools for a given hardware
- Not as efficient as hand-tuned programs, but quick production phase
Not reinventing the wheel... No NIH syndrome please!

Want to create your own tool?

- House-keeping and infrastructure in a compiler is a huge task
- Unreasonable to begin yet another new compiler project...
- Many academic Open Source projects are available...
- ...But customers need products 😊
- Integrate your ideas and developments in existing project
- ...or buy one if you can afford (ST with PGI...) 😊
- Some projects to consider
  - Old projects: gcc, PIPS... and many dead ones (SUIF...)
  - But new ones appear too: LLVM, RoseCompiler, Cetus...

Par4All

- Funding an initiative to industrialize Open Source tools
- Use source-to-source tool to be more target-independent
- PIPS is the first project to enter the Par4All initiative
Current PIPS usage

Developed for 23 years (...) @ Mines ParisTech & Télécom Bretagne, mainly

- 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]
- Compilation for hardware accelerators (Ter@PIX, SPoC, SIMD, FPGA, MPPA, P2012...) [FREIA, SCALOPES, SMECY]
- 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]
Par4All usage

- Generate from sequential C, Fortran & Scilab code
  - OpenMP for SMP
  - CUDA for nVidia GPU
  - OpenCL for GPU & ST Platform 2012 (on-going)
  - Code for various accelerators [SMECY], Kalray [SIMILAN]... (on-going)
  - SCMP task programs for SCMP machine from CEA and for... cloud computing 😊
Par4All ≡ PyPS scripting in the backstage (1)

- PIPS is a great tool-box to do source-to-source compilation
- ...but not really usable by λ end-user 😁
- → Development of Par4All
- Add a user compiler-like infrastructure
  - p4a script as simple as
    - p4a --openmp toto.c -o toto
    - p4a --cuda toto.c -o toto -lm
- Be multi-target
- Apply some adaptative transformations
- Up to now PIPS was scripted with a special shell-like language: tpips
- Not enough powerful (not a programming language)
- Develop a SWIG Python interface to PIPS phases and interface
Par4All $\equiv$ PyPS scripting in the backstage

- All the power of a widely spread real language
- Automate with introspection through the compilation flow
- Easy to add any glue, pre-/post-processing to generate target code

Overview

- Invoke PIPS transformations
  - With different recipes according to generated stuff
  - Special treatments on kernels...
- Compilation and linking infrastructure: can use gcc, icc, nvcc, nvcc+gcc, nvcc+icc
• House keeping code
• Fundamental: colorizing and filtering some PIPS output, running cursor... 😊
Coding rules

- Automatic parallelization is not magic
- Use abstract interpretation to « understand » programs
- Undecidable in the generic case (≈ halting problem)
- Quite easier for well written programs
- Develop a coding rule manual to help parallelization and... sequential quality!
  - Avoid useless pointers
  - Take advantage of C99 (arrays of non static size...)
  - Use higher-level C, do not linearize arrays...
  - Organize execution in cleaner loops expressing better parallelism
  - ...

- Prototype of coding rules report on-line on par4all.org
- The easy way... Already in PIPS
- Used to bootstrap the start-up with stage-0 investors 😊
- Indeed, we used only `bash-generated tpips` at this time (2008, no PyPS yet), but needed a lot of bug squashing on C support in PIPS...
OpenMP output sample

```c
!$omp parallel do private(I, K, X)
C multiply the two square matrices of ones
    DO  J = 1, N
0016
!$omp parallel do private(K, X)
    DO  I = 1, N
0017
        X = 0
0018
!$omp parallel do reduction(+:X)
    DO  K = 1, N
0019
        X = X+A(I,K)*B(K,J)
0020
    ENDDO
!$omp end parallel do
    C(I,J) = X
0022
    ENDDO
!$omp end parallel do
    ENDDO
!$omp end parallel do
```

Programming environments for big data processing on modern parallel architectures

HPC Project

Ronan KERYELL
Outline

1. Hardware architectures
   - Classical multicores
   - GPU
   - MP-SoC underworlds

2. Software environments
   - Programming challenges
   - Multicores
   - GPU
   - Application libraries

3. Par4All
   - GPU code generation
   - Code generation for SCMP

4. Conclusion
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
Scilab & Matlab

- Languages used for simulations, data analytics, pricing...
- Scilab/Matlab input: *sequential* or array syntax
- Compilation to C code
  - Side effect of MediaGPU ANR project...
  - Our COLD compiler is *not* Open Source
  - There is such Open Source compiler from hArtes European project written in... Scilab 😊
- Parallelization of the generated C code
- Use parallel runtime too
- Type inference to guess (crazy 😊) semantics
  - Heuristic: first encountered type is forever
- May get speedup >> 1000 😊
- Wild Cruncher product from HPC Project: x86+GPU appliance with nice interface
  - Scilab — mathematical model & simulation
  - Par4All — automatic parallelization
  - //Geometry — polynomial-based 3D rendering & modelling
Geographical application for data integration: library to compute neighbourhood population potential with scale control

WildNode with 2 Intel Xeon X5670 @ 2.93GHz (12 cores) and a nVidia Tesla C2050 (Fermi), Linux/Ubuntu 10.04, gcc 4.4.3, CUDA 3.1

- Sequential execution time on CPU: 30.355s
- OpenMP parallel execution time on CPUs: 3.859s, speed-up: 7.87
- CUDA parallel execution time on GPU: 0.441s, speed-up: 68.8

With single precision on a HP EliteBook 8730w laptop (with an Intel Core2 Extreme Q9300 @ 2.53GHz (4 cores) and a nVidia GPU Quadro FX 3700M (16 multiprocessors, 128 cores, architecture 1.1)) with Linux/Debian/sid, gcc 4.4.5, CUDA 3.1:

- Sequential execution time on CPU: 34.7s
- OpenMP parallel execution time on CPUs: 13.7s, speed-up: 2.53
- OpenMP emulation of GPU on CPUs: 9.7s, speed-up: 3.6
- CUDA parallel execution time on GPU: 1.57s, speed-up: 24.2
Original main C kernel:

```c
void run(data_t xmin, data_t ymin, data_t xmax, data_t ymax, data_t step, data_t range,
          town pt[rangex][rangey], town t[nb])
{
    size_t i,j,k;

    fprintf(stderr,"begin\_computation\_...\n");

    for (i=0;i<rangex;i++)
        for (j=0;j<rangey;j++) {
            pt[i][j].latitude = (xmin + step * i) * 180 / M_PI;
            pt[i][j].longitude = (ymin + step * j) * 180 / M_PI;
            pt[i][j].stock = 0.;
            for (k=0;k<nb;k++) {
                data_t tmp = 6368.*acos(cos(xmin+step*i)*cos(t[k].latitude)
                                      *cos((ymin+step*j)-t[k].longitude)
                                      +sin(xmin+step*i)*sin(t[k].latitude));
                if (tmp < range)
                    pt[i][j].stock += t[k].stock / (1 + tmp);
            }
        }
    fprintf(stderr,"end\_computation\_...\n");
}
```
Example given in par4all.org distribution
OpenMP code:

```c
void run(data_t xmin, data_t ymin, data_t xmax, data_t ymax, data_t step, data_t range, town pt[290][299], town t[2878]) {
    size_t i, j, k;

    fprintf(stderr, "begin\_computation\_...\n");

    #pragma omp parallel for private(k, j)
    for (i = 0; i <= 289; i += 1)
    for (j = 0; j <= 298; j += 1) {
        pt[i][j].latitude = (xmin + step * i)*180/3.14159265358979323846;
        pt[i][j].longitude = (ymin + step * j)*180/3.14159265358979323846;
        pt[i][j].stock = 0.;
        for (k = 0; k <= 2877; k += 1) {
            data_t tmp = 6368. * acos(cos(xmin + step * i)*cos(t[k].latitude)*cos(ymin + step * j-t[k].longitude) + sin(xmin + step * i)*sin(t[k].latitude));
            if (tmp < range) pt[i][j].stock += t[k].stock /(1+ tmp);
        }
    }
    fprintf(stderr, "end\_computation\_...\n");
}

void display(town pt[290][299]) {
}
size_t i, j;
for (i = 0; i <= 289; i += 1) {
    for (j = 0; j <= 298; j += 1)
        printf("%lf %lf %lf \n", pt[i][j].latitude, pt[i][j].longitude, pt[i][j].stock);
    printf("\n");
}

Programming environments for big data processing on modern parallel architectures
Generated GPU code:

```c
void run(data_t xmin, data_t ymin, data_t xmax, data_t ymax, data_t step, data_t range,
          town pt[290][299], town t[2878])
{
    size_t i, j, k;
    //PIPS generated variable
    town (*P_0)[2878] = (town (*)[2878]) 0, (*P_1)[290][299] = (town (*)[290][299]) 0;
    fprintf(stderr, "begin\ncomputation\n...\n\n");
    P4A_accel_malloc(&P_1, sizeof(town[290][299])-1+1);
    P4A_accel_malloc(&P_0, sizeof(town[2878])-1+1);
    P4A_copy_to_accel(pt, *P_1, sizeof(town[290][299])-1+1);
    P4A_copy_to_accel(t, *P_0, sizeof(town[2878])-1+1);
    p4a_kernel_launcher_0(*P_1, range, step, *P_0, xmin, ymin);
    P4A_copy_from_accel(pt, *P_1, sizeof(town[290][299])-1+1);
    P4A_accel_free(*P_1);
    P4A_accel_free(*P_0);
    fprintf(stderr, "end\ncomputation\n...\n\n");
}

void p4a_kernel_launcher_0(town pt[290][299], data_t range, data_t step, to
                           data_t xmin, data_t ymin)
```

- Programming environments for big data processing on modern parallel architectures
- HPC Project
- Ronan Keryell
```c
{ //PIPS generated variable
    size_t i, j, k;
    P4A_call_accel_kernel_2d(p4a_kernel_wrapper_0, 290, 299, i, j, pt, range, step, t, xmin, ymin);
}

P4A_accel_kernel_wrapper void p4a_kernel_wrapper_0(size_t i, size_t j, town pt[290][299],
                                                   data_t range, data_t step, town t[2878], data_t xmin, data_t ymin)
{
   // Index has been replaced by P4A_vp_0:
   i = P4A_vp_0;
   // Index has been replaced by P4A_vp_1:
   j = P4A_vp_1;
   // Loop nest P4A end
   p4a_kernel_0(i, j, &pt[0][0], range, step, &t[0], xmin, ymin);
}

P4A_accel_kernel void p4a_kernel_0(size_t i, size_t j, town *pt, data_t range,
                                      data_t step, town *t, data_t xmin, data_t ymin)
{
   //PIPS generated variable
   size_t k;
   // Loop nest P4A end
```
```c
if (i<=289&&j<=298) {
    pt[299*i+j].latitude = (xmin+step*i)*180/3.14159265358979323846;
    pt[299*i+j].longitude = (ymin+step*j)*180/3.14159265358979323846;
    pt[299*i+j].stock = 0.;
    for (k = 0; k <= 2877; k += 1) {
        data_t tmp = 6368.*acos(cos(xmin+step*i)*cos((*(t+k)).latitude)*cos(-(*(t+k)).longitude)+sin(xmin+step*i)*sin((*(t+k)).latitude));
        if (tmp<range)
            pt[299*i+j].stock += t[k].stock/(1+tmp);
    }
}
```
Stars-PM

- *Particle-Mesh* N-body cosmological simulation
- C code from Observatoire Astronomique de Strasbourg
- Use FFT 3D
- Example given in par4all.org distribution
void iteration(coord pos[NP][NP][NP],
              coord vel[NP][NP][NP],
              float dens[NP][NP][NP],
              int data[NP][NP][NP],
              int histo[NP][NP][NP]) {
    /* Split space into regular 3D grid: */
    discretisation(pos, data);
    /* Compute density on the grid: */
    histogram(data, histo);
    /* Compute attraction potential in Fourier’s space: */
    potential(histo, dens);
    /* Compute in each dimension the resulting forces and integrate the acceleration to update the speeds: */
    forcex(dens, force);
    updatevel(vel, force, data, 0, dt);
    forcey(dens, force);
    updatevel(vel, force, data, 1, dt);
    forcez(dens, force);
    updatevel(vel, force, data, 2, dt);
    /* Move the particles: */
    updatepos(pos, vel);
}
Stars-PM & Jacobi results with p4a 1.1.2

- 2 Xeon Nehalem X5670 (12 cores @ 2.93 GHz)
- 1 GPU nVidia Tesla C2050 CUDA 3.2
- Automatic call to CuFFT instead of FFTW (stubs...)
- 150 iterations of Stars-PM

<table>
<thead>
<tr>
<th>Execution time</th>
<th></th>
<th>p4a</th>
<th>Simulation Cosmo.</th>
<th>Jacobi</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>32^3</td>
<td>64^3</td>
<td>128^3</td>
</tr>
<tr>
<td>Sequential</td>
<td>gcc -03</td>
<td>0.68</td>
<td>6.30</td>
<td>98.4</td>
</tr>
<tr>
<td>OpenMP 6 threads</td>
<td>--openmp</td>
<td>0.16</td>
<td>1.28</td>
<td>16.6</td>
</tr>
<tr>
<td>CUDA base</td>
<td>--cuda</td>
<td>0.88</td>
<td>5.21</td>
<td>31.4</td>
</tr>
<tr>
<td>Optim. comm. 1.1</td>
<td>--cuda</td>
<td>0.20</td>
<td>1.17</td>
<td>8.9</td>
</tr>
<tr>
<td>Reduction Optim. 1.1.2</td>
<td>--com-opt.</td>
<td>0.10</td>
<td>0.32</td>
<td>2.1</td>
</tr>
<tr>
<td>Manual optim.</td>
<td>gcc -03</td>
<td>0.05</td>
<td>0.26</td>
<td>1.8</td>
</tr>
</tbody>
</table>

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

1. Hardware architectures
   - Classical multicores
   - GPU
   - MP-SoC underworlds

2. Software environments
   - Programming challenges
   - Multicores
   - GPU
   - Application libraries

3. Par4All
   - GPU code generation
   - Code generation for SCMP

4. Conclusion
Future revolutions

Software radio, cognitive radio, passive radar, compressed sensing...

- Embedded accelerator developed at French CEA
  - Task graph oriented parallel multiprocessor
  - Hardware task graph scheduler
  - Synchronizations
  - Communication through memory page sharing

- Generating code from THALES (TCF) GSM sensing application in SCALOPES European project

- Reuse output of PIPS GPU phases + specific phases
  - SCMP code with tasks
  - SCMP task descriptor files

- Adapted Par4All Accel run-time

Programing environments for big data processing on modern parallel architectures
In general case, different tasks can produce data in unpredictable way: use helper data server tasks to deal with coherency when several producers
```c
int main() {
    int i, t, a[20], b[20];
    for (t = 0; t <= 99; t += 1) {
        [...] 
    }
    return (0);
}

int main() {
P4A_scmp_reset();
    int i, t, a[20], b[20];
    for (t = 0; t <= 99; t += 1) {
        [...] 
    }
    return (ev_T004);
}
```

### SCMP task code (before/after)

**SCMP task code (before)\**

```c
int main () {
    int i, t, a[20], b[20];
    for (t = 0; t < 100; t++) {
        kernel_tasks_1:
            for (i = 0; i < 10; i++)
                a[i] = i + t;
        kernel_tasks_2:
            for (i = 10; i < 20; i++)
                a[i] = 2 * i + t;
        kernel_tasks_3:
            for (i = 10; i < 20; i++)
                printf("a[%d]/uni2423=/uni2423%d
", i, a[i]);
    }
    return (0);
}
```

**SCMP task code (after)\**

```c
int main() {
P4A_scmp_reset();
    int i, t, a[20], b[20];
    for (t = 0; t < 100; t++) {
        [...]
    }
    //PIPS generated variable
    int (*P4A__a__1)[10] = (int (*)[10]) 0;
P4A_scmp_malloc((void **) &P4A__a__1,
        sizeof(int)*10, P4A__a__1_id,
P4A__a__1_prod_p || P4A__a__1_cons_p, P4A__a__1_id);
    if (scmp_task_2_p)
        for (i = 10; i <= 19; i += 1)
            (*P4A__a__1)[i - 10] = 2 * i + t;
P4A_copy_fromaccel_1d(sizeof(int), 20, 10,
P4A_sesam_server_a_p ? &a[0] : NULL, *P4A__a__1,
P4A__a__1_prod_p || P4A__a__1_cons_p);
P4A_scmp_dealloc(P4A__a__1, P4A__a__1_id,
P4A__a__1_prod_p || P4A__a__1_cons_p, P4A__a__1_id);
    return (0);
}
```

---

**Programming environments for big data processing on modern parallel architectures**

**HPC Project**

**Ronan KERYELL**

96 / 101
Performance of GSM sensing on SCMP

- Speed-up on 4 PE SCMP:
  - ×2.35 with manual parallelization by SCMP team
  - ×1.86 with automatic Par4All parallelization

- Still big memory overhead
- To optimize...
- Use these techniques to generate automate cloud-ification 😊
Performance of GSM sensing on SCMP

- Speed-up on 4 PE SCMP:
  - ×2.35 with manual parallelization by SCMP team
  - ×1.86 with automatic Par4All parallelization
- Still big memory overhead
- To optimize...
- Use these techniques to generate automate cloud-ification 😊
Outline

1. Hardware architectures
   - Classical multicores
   - GPU
   - MP-SoC underworlds

2. Software environments
   - Programming challenges
   - Multicores
   - GPU
   - Application libraries

3. Par4All
   - GPU code generation
   - Code generation for SCMP

4. Conclusion
French advantage: Saint (*Holly*) Cloud localization

Programming environments for big data processing on modern parallel architectures
Conclusion

Saint Cloud gatekeeper & massive virtual I/O
Conclusion

- Quantum change in data amount to process
- Getting out of old tracks for innovative solutions
- Efficiency future will be more and more heterogeneous 😊
  - Processors everywhere
  - Smart routers with on-the-fly processing
- Hardware is no longer oblivious Challenge of plain object oriented modeling 😞
- Low latency is difficult (high-frequency trading...)
- Need expertise in hardware, software & applications...
- No unique software environment 😞
- Opportunity to modernize legacy applications
- Good trade-off between efficiency and portability in some areas: OpenCL
- Automatic tools such as Par4All can reduce time-to-market to generate // code to a given target (from low- to high-level)