Par4All

From Sequential Applications to Heterogeneous Parallel Computing

Ronan KERYELL (Ronan.Keryell@wild-systems.com)
Chief Scientist

Wild Systems
—
5201 Great America Parkway #320
Santa Clara, CA 95054, USA

04/23/2012

Some French archæology (I)

- First integrated CRT terminal controller in the world designed in Paris at École Normale Supérieure, 1975
  - Mostly done by Jean GASTINEL with real manual floor-planning on... student room floor!
  - EF9364, 6μm polySi-grid NMOS
    google.com/patents/US4328557.pdf
- The sequel: EF9365+ 2D graphics controller with BRESENHAM's algorithm & parallel workstations
- Used as a basis for the Minitel display controller
  - Multics concepts for real
  - Millions of videotext terminals with modem deployed in France and other countries during 1980-1990's (7M in France, 1995)
  - As a student, launched a start-up to do parallel Minitel servers
    - Atari ST with MIDI as a LAN (!), 1986
    - 128 users/Atari to compete with ATT 3B15 standard at that time

Some French archæology (II)

- There is a Minitel exhibited in the San Jose airport! ☺

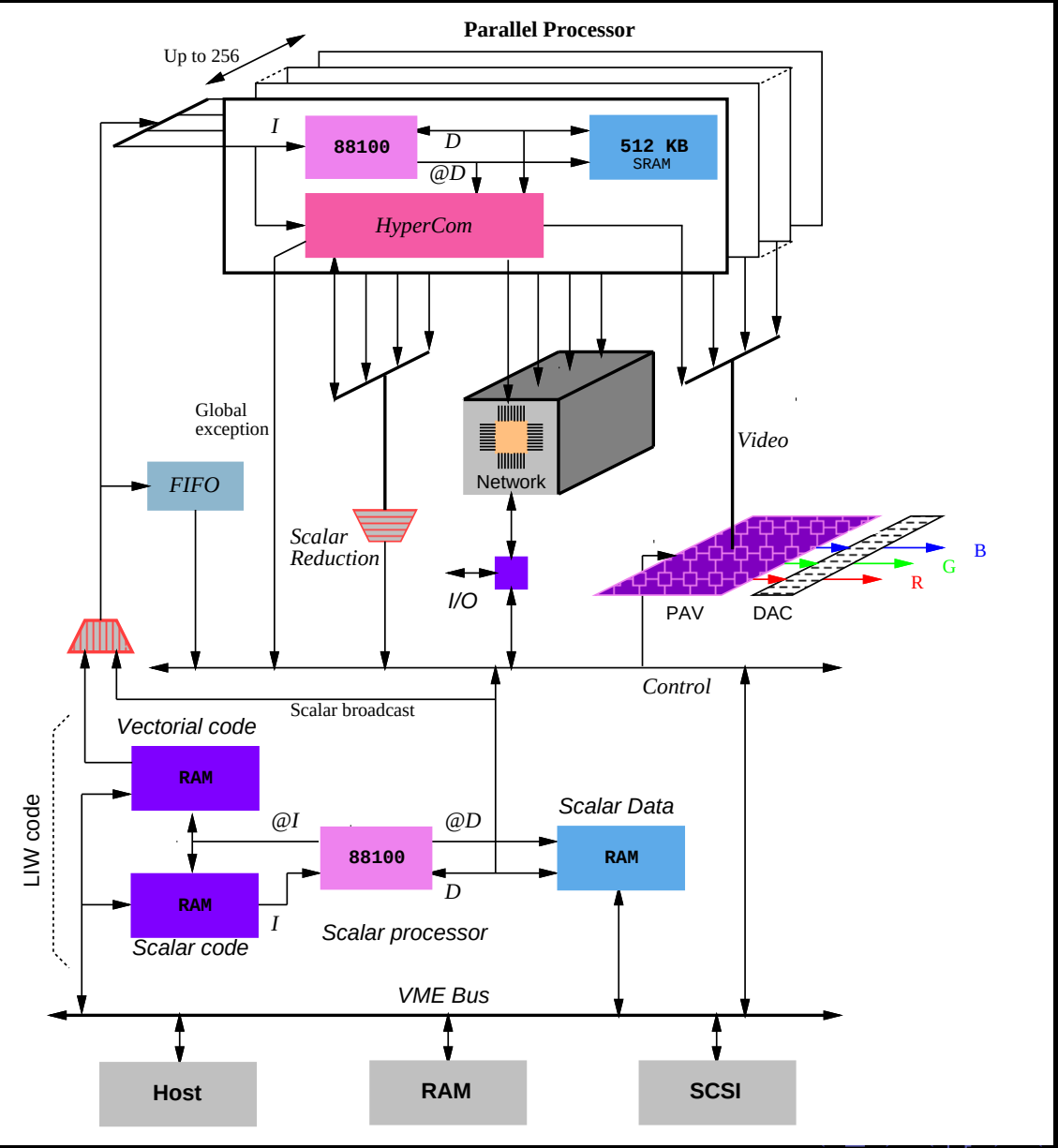
Eduardo Kac (Minitel), Small Wonders, 2010: The Art and Technology Network, Norman Mineta San Jose International Airport

Some French archæology *(III)*

- 1986: half of the team moved to Xerox PARC to design parallel computers (Dragon) and later in various other companies (Sun SS1000+...)
- 1982–1990 @ LI/ENS: FLIP graphics pipeline slice processors for scalable GPU
- Image Synthesis team @ LI/ENS was designing advanced method such as radiosity+ray tracing (CIL)

⇝ Current architecture was not flexible enough ☹

POMP & PompC @ LI/ENS 1987–1992



HyperParallel Technologies (1992–1998)

- Parallel computer
- DEC Alpha 21064
- FPGA-based 3D-torus network
- HyperC (follow-up of PompC @ LI/ENS Ulm)
  - PGAS (Partitioned Global Address Space) language
  - An ancestor of UPC...

Quite simple business model

- Customers need just to rewrite all their code in HyperC ☺
- Difficult entry cost... ☹

- Killed by niche market + side effect of French bank scandal... ☹
- American subsidiary with dataparallel analytics application acquired by Yahoo! in 1998, $8M
- Closed technology ⇝ lost for customers and... founders ☹
- But some concepts still in use in modern GPU: Ronan KERYELL & Nicolas PARIS. « Activity Counter : New Optimization for the Dynamic Scheduling of SIMD Control Flow. » ICPP 1993

HyperParallel Technologies (1992–1998)

- Parallel computer
- DEC Alpha 21064
- FPGA-based 3D-torus network
- HyperC (follow-up of PompC @ LI/ENS Ulm)
  - PGAS (Partitioned Global Address Space) language
  - An ancestor of UPC...

Quite simple business model

- Customers need just to rewrite all their code in HyperC ☺
- Difficult entry cost... ☹

- Killed by niche market + side effect of French bank scandal... ☹
- American subsidiary with dataparallel analytics application acquired by Yahoo! in 1998, $8M
- Closed technology ⇝ lost for customers and... founders ☹
- But some concepts still in use in modern GPU: Ronan KERYELL & Nicolas PARIS. « Activity Counter : New Optimization for the Dynamic Scheduling of SIMD Control Flow. » ICPP 1993

Parallelism is the only way to go

Just wait for $\mathcal{O}(1)$ years...

Outline

HPC Project emergence

- ≈ 2006 : power consumption
  - ⇝ Multicores and programmable GPU almost mainstream!

Time to be back in parallelism!

Yet another start-up... ☺

- Friends that met ≈ 1990 at the French Parallel Computing military lab SEH/ETCA
- Later became researchers in Computer Science, CINES director and ex-CEA/DAM (≈ ASCI in US), venture capital and more: ex-CEO of Thales Computer, HP marketing...
- HPC Project launched in December 2007
- Now ≈ 35 colleagues in France (Montpellier, Meudon), Canada (Montréal with Parallel Geometry) & USA (Wild Systems in Santa Clara, CA)
- Just closing our 3rd fund raising

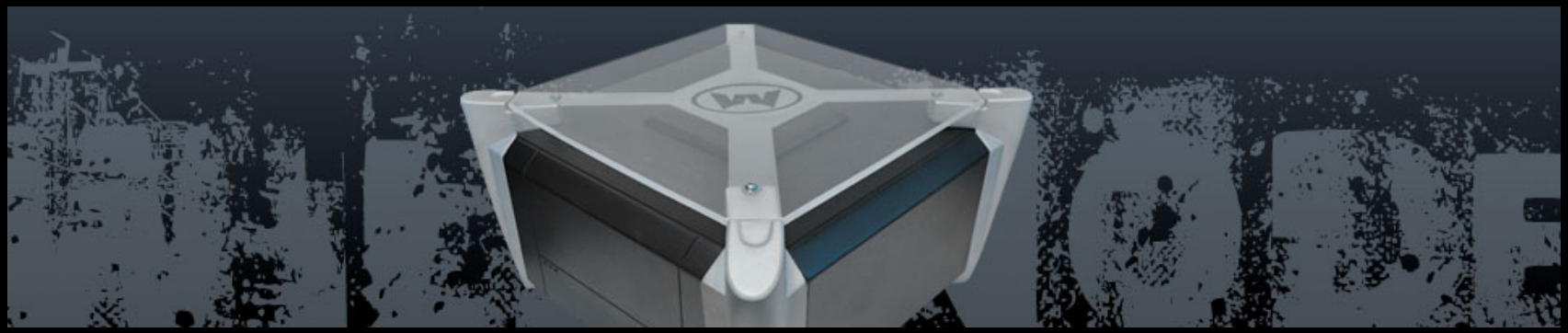
HPC Project business

Wide expertise in parallel computing & high-end simulation

- Engineering services (application development for parallel & embedded systems)
- Parallel simulator architectures (military training...)
- Libraries for dense & sparse linear algebra on multiple heterogeneous accelerators
- Parallelizing tools for Scilab, C & Fortran to OpenMP, CUDA & OpenCL: Par4All
- Professional training (parallel programming, OpenMP, MPI, TBB, CUDA, OpenCL...)

Wild Node

- Agreements with some ISV to provide optimized-application-in-the-box
- WildNode hardware desktop accelerator
  - Low noise for in-office operation
  - *x*86 manycore
  - nVidia Tesla GPU Computing
  - Linux & Windows



- Wild Hive
  - Aggregate 2-4 nodes with 2 possible memory views

Outline

Expressing/finding parallelism ?

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

Automatic parallelization (I)

- Major research failure from the past...
- ... We used to work on automatic parallelization & HPF Fortran compilation
- Untractable in the general case ☹
- But automatic parallelization technology widely used locally in main compilers
- Bad sequential programs? GIGO: Garbage In-Garbage Out...
- To use #pragma, // languages or classes: cleaner sequential program or algorithm first...

Automatic parallelization (II)

Agent SMITH:

Never send a human to do a machine's job.

In *Matrix* (Andy & Larry WACHOWSKI, 1999)

Automatic parallelization (III)

- ... 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 (I)

- 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`
- No `exit()`, `longjump()`, `setcontext()`...
- Data structures
  - Pointers
    - Do not use pointer arithmetics

Basic Par4All coding rules for good parallelization (II)

- Arrays
  - PIPS uses integer polyhedron lattice in analysis
  - Do not use linearized arrays
  - Use affine reference in parallelizable code

```
// Good:
a[2*i-3+m][3*i-j+6*n]
// Bad (polynomial):
a[2*i*j][m*n-i+j]
```

- Do not use recursion
- Prototype of coding rules report on-line on `par4all.org`

Outline

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

```
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);
}
```

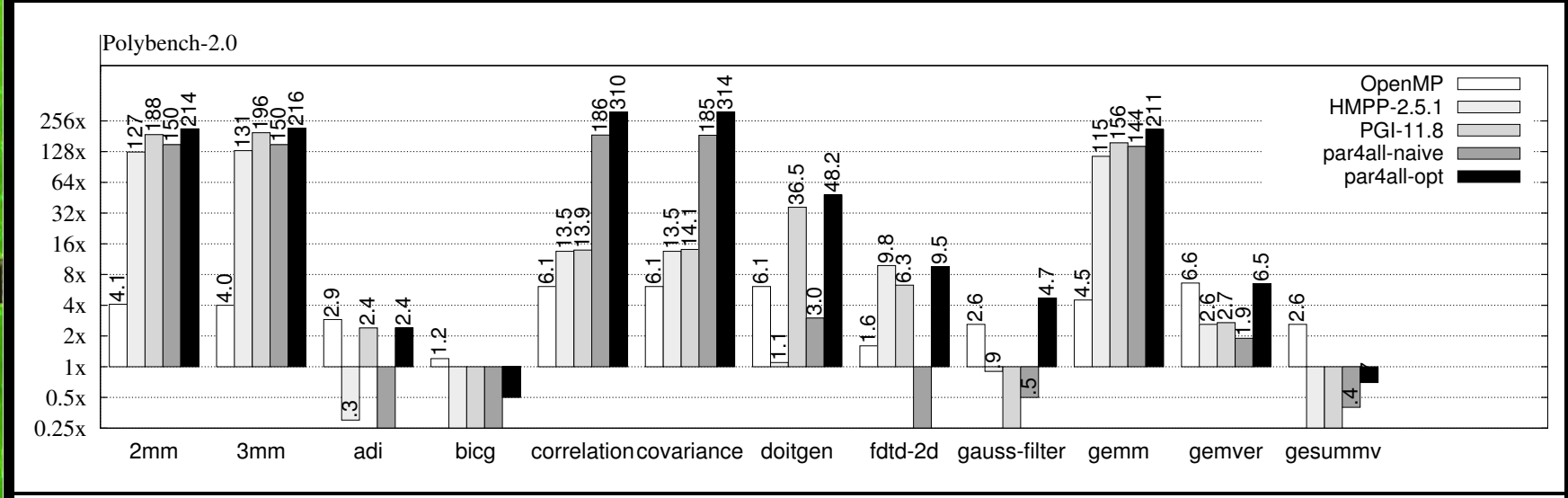
Stars-PM & Jacobi results

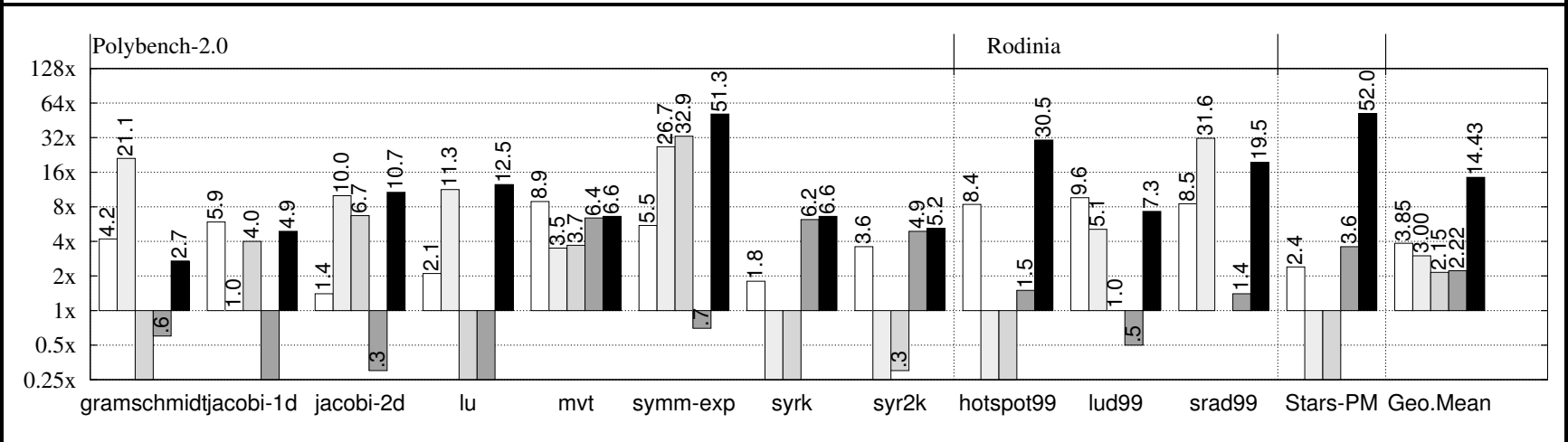
- 2 Xeon Nehalem X5670 (12 cores @ 2,93 GHz)
- 1 GPU nVidia Tesla C2050
- Automatic call to CuFFT instead of FFTW
- Time and speed-up for 150 iterations of Stars-PM

Speed-up	p4a	Simulation Cosmo. $128 \times 128 \times 128$	Jacobi
Sequential (time in s)	(gcc -O3)	98.4 s	24.5 s
OpenMP *6 threads*	--openmp	5.9	1.78
CUDA base	--cuda	3.13	0.36
Optim. comm.	+ --com-optim	11	3.8
Reduction Optim.	+ --atomic	46.9	6.4
Manual optim.	(gcc -O3)	54.7	

Benchmark results

With Par4All 1.2, CUDA 4.0, WildNode 2 Xeon Nehalem X5670 (12 cores @ 2.93 GHz) with nVidia C2050. [LCPC'2011]





From `par4all.org` distribution, in `examples/Benchmarks`

Outline

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

- Scilab/Matlab input : *sequential* or array syntax
- Compilation to C code with our own compiler
- Parallelization of the generated C code
- Type inference to guess (crazy ☹) semantics
  - Heuristic: first encountered type is forever
- Speedup $>$ 1000 in some cases ☺
- Wild Cruncher: *x*86+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 IDE

Applications Places System Wed Jun 29, 3:56 PM ronan

WildCruncher

Simulation: Mandelbrot.sce | Workspace: demo

Mandelbrot.sce (/home/wildcruncher/WildCruncher/WCM_DATA/Demo/Mandelbrot/Mandelbrot.sce)

```
// Compute the image
nmax = 300;
xmin = 0.2675;
xmax = 0.2685;
ymin = 0.591;
ymax = 0.592;
xsize =1000;
ysize =1000;

xstep=(xmax-xmin)/(xsize-1);
ystep=(ymax-ymin)/(ysize-1);

result=zeros(xsize,ysize);

tic();
for i=1:xsize
  for j=1:ysize
    x = xmin+i*xstep;
    y = ymin+j*ystep;
    x0=x;
    y0=y;
    k=0;
    while( (x*x+y*y) < 4 & k< nmax)
      xtemp=x*x - y*y + x0;
      y=2*x*y + y0;
      x=xtemp;
      k=k+1;
```

Console

```
 0.585938
   Executing OpenMP version of 'Mandelbrot.sce'
 Shared archive loaded.
Link done.
 ans =
 0.05637
fact =
 0.585938
-->
```

Information

```
p4a: PIPS: user warning in c_convex_effects_on_formal_parameter_backward_translation: Cast in actual parameter -> anywhere effect
p4a: PIPS: user warning in call_proper_reduction_p: not handeled case, no reduction will be detected
Compiling Scilab code "/home/wildcruncher/WildCruncher/WCM_DATA/Demo/Mandelbrot/Mandelbrot.sce" to C code "/home/wildcruncher/WildCruncher/W
Filtering code in "/home/wildcruncher/WildCruncher/WCM_DATA/Demo/Mandelbrot/p4a_Mandelbrot.sce/Mandelbrot.cold.c"
Parallelizing code "/home/wildcruncher/WildCruncher/WCM_DATA/Demo/Mandelbrot/p4a_Mandelbrot.sce/p4a/Mandelbrot.cold-pre-p4a.c" with p4a
Compiling final code "/home/wildcruncher/WildCruncher/WCM_DATA/Demo/Mandelbrot/p4a_Mandelbrot.sce/Mandelbrot.cold.p4a.c"
End compilation
```

Graphic window number 0

[wildcruncher@WildCru... [Terminal] WildCruncher

Outline

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 (I)

- Too difficult to start yet another compiler project...
- PIPS (Interprocedural Parallelizer of Scientific Programs): Open Source project from MINES ParisTech... 23-year old! ☺
- ≈ 456 KLOC according to David A. Wheeler's `SLOCCount`
- ≈ 300 phases (parsers, analyzers, transformations, optimizers, parallelizers, code generators, pretty-printers...) that can be combined for the right purpose
- **Abstract interpretation**
- **Polytope lattice** (**sparse** linear algebra) used for semantics analysis, transformations, code generation... with **approximations** to deal with big programs
- One of the project that introduced polytope model-based compilation

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]

Outlining (I)

Parallel code ⇝ Kernel code on GPU

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

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

Outlining (II)

- After:

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

From array regions to GPU memory allocation (I)

Example

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

- Memory accesses are summed up by inference for each statement as *regions* for array accesses: integer polytope lattice

```
// <h_A[PHI1][PHI2]-W-EXACT-{0<=PHI1, PHI2+1<=n, PHI1<=PHI2}>
  for(i = 0; i <= n-1; i += 1)
// <h_A[PHI1][PHI2]-W-EXACT-{PHI1==i, i<=PHI2, PHI2+1<=n, 0<=i}>
    for(j = i; j <= n-1; j += 1)
// <h_A[PHI1][PHI2]-W-EXACT-{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

```
cudaMemcpy(accel_address, host_address,
           size, cudaMemcpyHostToDevice)
```

  - copy-out

```
cudaMemcpy(host_address, accel_address,
           size, cudaMemcpyDeviceToHost)
```

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 ⚠

From preconditions to iteration clamping (II)

- So we need to generate code such as

```
void p4a_kernel_wrapper_0(int k, int l,...)
{
  k = blockIdx.x*blockDim.x + threadIdx.x;
  l = blockIdx.y*blockDim.y + threadIdx.y;
  if (k >= 0 && k <= M - 1 && l >= 0 && l <= M - 1)
    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:

```
// 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 ≡ 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

- Many heterogeneous targets and language
- Difficult to represent them internally
- ⇝ Common abstraction layer
  - CUDA
  - OpenCL
  - Ter@pix SIMD processor
  - MCA API (MultiCore Association)
- Implementation not that far from OpenCL C++ presented last time by Ben CASTER
- Also a pure C version for non C++ target
- Can be used to simplify manual programming too (OpenCL...)
  - Manual radar electromagnetic simulation code @TB
  - Only 1 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...)

`http://download.par4all.org/doc/Par4All_Accel_runtime/graph`

Hyantes (I)

- Geographical application: library to compute neighbourhood population potential with scale control
- Example given in `par4all.org` distribution

Original main C kernel:

```
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");
}
```

Hyantes (II)

OpenMP code:

```
void run(data_t xmin, data_t ymin, data_t xmax, data_t ymax, data_t step, d
{
   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
            if (tmp<range)
               pt[i][j].stock += t[k].stock/(1+tmp);
         }
      }
   fprintf(stderr, "end computation ...\n");
}
void display(town pt[290][299])
{
```

Hyantes (III)

```
    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
        printf("\n");
    }
}
```

Hyantes (IV)

Generated GPU code:

```
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 computation ...\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 computation ...\n");
}

void p4a_kernel_launcher_0(town pt[290][299], data_t range, data_t step, town t[2878],
  data_t xmin, data_t ymin)
{
   //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],
```

Hyantes (V)

```
  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
   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(ymin+step*j
               -(*(t+k)).longitude)+sin(xmin+step*i)*sin((*(t+k)).latitude));
         if (tmp<range)
            pt[299*i+j].stock += t[k].stock/(1+tmp);
      }
   }
}
```

Outline

Conclusion

- No tool or language will solve all the issues...
- Par4All target:
  - Scientific programming
  - Non GPU-specialist programmers
  - Time-to-market instead of maximum performance
- Coding rules to help cleaning programs and parallelization
  - Take a positive attitude... Parallelization is a good opportunity for deep cleaning (refactoring, modernization...) ⇝ improve also the original code
- Rely on open standards
- Open Source for community network effect
- ⚠ Entry cost & ⚠ ⚠ ⚠ Exit cost! ☹
  - Do not loose control on *your* code and *your* data !
- We are hiring C/C++11 programmers
- Moving good ideas to Clang(/LLVM) to tackle C++
- See you next month on our booth at GTC'2012

Conclusion

- No tool or language will solve all the issues...
- Par4All target:
  - Scientific programming
  - Non GPU-specialist programmers
  - Time-to-market instead of maximum performance
- Coding rules to help cleaning programs and parallelization
  - Take a positive attitude... Parallelization is a good opportunity for deep cleaning (refactoring, modernization...) ⇝ improve also the original code
- Rely on open standards
- Open Source for community network effect
- ⚠ Entry cost & ⚠ ⚠ ⚠ Exit cost! ☹
  - Do not loose control on *your* code and *your* data !
- We are hiring C/C++11 programmers
- Moving good ideas to Clang(/LLVM) to tackle C++
- See you next month on our booth at GTC'2012