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

- 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

- 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

- 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 😞

- 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 😞

- 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 😞
Parallelism is the only way to go

Just wait for $O(1)$ years...
Outline

1. HPC Project
2. Par4All
3. Results
4. Scilab to OpenMP, CUDA & OpenCL
5. Par4All internals
6. Conclusion
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
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...)
Agreements with some ISV to provide optimized-application-in-the-box

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

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

- 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...
Agent SMITH: *Never send a human to do a machine’s job.*

In *Matrix* (Andy & Larry WACHOWSKI, 1999)
... 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. Same constraints as for-loop accepted in OpenMP standard
   
   ```
   for ([int] init-expr; var relational-op b; incr-expr)
   statement
   ```

2. Increment and bounds: integer expressions, loop-invariant

3. `relational-op` only `<`, `<=`, `>=`, `>`

4. Do not modify loop index inside loop body

5. Do not use `assert()` or compile with `-DNDEBUG` inside a loop. Assert has potential exit effect

6. No `goto` outside the loop, `break`, `continue`

7. No `exit()`, `longjump()`, `setcontext()`...

8. 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
Results

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: */
    forcelx(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);
}
Results

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

<table>
<thead>
<tr>
<th>Speed-up</th>
<th>p4a (gcc -O3)</th>
<th>Simulation Cosmo. 128 × 128 × 128</th>
<th>Jacobi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential (time in s)</td>
<td>(gcc -O3) --openmp</td>
<td>98.4 s</td>
<td>24.5 s</td>
</tr>
<tr>
<td>OpenMP 6 threads</td>
<td></td>
<td>5.9</td>
<td>1.78</td>
</tr>
<tr>
<td>CUDA base</td>
<td>--cuda</td>
<td>3.13</td>
<td>0.36</td>
</tr>
<tr>
<td>Optim. comm.</td>
<td>+ --com-optim</td>
<td>11</td>
<td>3.8</td>
</tr>
<tr>
<td>Reduction Optim.</td>
<td>+ --atomic</td>
<td>46.9</td>
<td>6.4</td>
</tr>
<tr>
<td>Manual optim.</td>
<td>(gcc -O3)</td>
<td>54.7</td>
<td></td>
</tr>
</tbody>
</table>

Wild Systems

R. Keryell
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

1. HPC Project
2. Par4All
3. Results
4. Scilab to OpenMP, CUDA & OpenCL
5. Par4All internals
6. Conclusion
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: 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 Systems

R. Keryell
Wild Cruncher — Scilab parallelization IDE
Outline

1. HPC Project
2. Par4All
3. Results
4. Scilab to OpenMP, CUDA & OpenCL
5. Par4All internals
6. Conclusion
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
Par4All internals

Rely on PIPS

- 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]
Parallel code $\rightsquigarrow$ Kernel code on GPU

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

- Before:

```c
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]);
    }
```
After:

```c
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

Example

```plaintext
define PHI1, PHI2

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

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

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

  ```
  cudaMemcpy (host_address, accel_address,
              size, cudaMemcpyDeviceToHost)
  ```
From preconditions to iteration clamping

- 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.

Par4All internals

Wild Systems

R. Keryell
From preconditions to iteration clamping

- So we need to generate code such as

```c
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:

```c
// P(i, j, k, l) {0<=k,  k<=63, 0<=l,  l<=63}
```

- Guard ≡ directly translation in C of preconditions on loop indices that are GPU thread indices
Optimized reduction generation

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

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");
}
```
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])
{
```
```c
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");
}
```
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],
...
```c
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

1. HPC Project
2. Par4All
3. Results
4. Scilab to OpenMP, CUDA & OpenCL
5. Par4All internals
6. Conclusion
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
## Table of Content

<table>
<thead>
<tr>
<th>Page</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>HPC Project</td>
</tr>
<tr>
<td>2</td>
<td>Par4All</td>
</tr>
<tr>
<td>3</td>
<td>Results</td>
</tr>
<tr>
<td>4</td>
<td>Scilab to OpenMP, CUDA &amp; OpenCL</td>
</tr>
<tr>
<td>5</td>
<td>Par4All internals</td>
</tr>
<tr>
<td>6</td>
<td>Conclusion</td>
</tr>
</tbody>
</table>

### HPC Project
- Outline
- HPC Project emergence
- HPC Project business
- Wild Node

### Par4All
- Outline
- Expressing/finding parallelism?
- Automatic parallelization
- Basic Par4All coding rules for good parallelization

### Results
- Outline
- Stars-PM
- Stars-PM time step
- Stars-PM & Jacobi results
- Benchmark results

### Scilab to OpenMP, CUDA & OpenCL

### Par4All internals
- Outline
- Basic GPU execution model
- Rely on PIPS
- Current PIPS usage
- Outlining
- From array regions to GPU memory allocation
- Communication generation
- From preconditions to iteration clamping
- Optimized reduction generation
- Communication optimization
- Loop fusion
- Par4All Accel Runtime
- Hyantes

### Conclusion
- Outline
- Conclusion
- Conclusion

You are here!