Par4All
From Convex Array Regions to Heterogeneous Computing

Mehdi Amini, Béatrice Creusillet, Stéphanie Even, Ronan Keryell, Onig Goubier, Serge Guelton, Janice Onanian McMahon, François Xavier Pasquier, Grégoire Péan, Pierre Villalon
Par4All project: automatic source-to-source parallelization for heterogeneous targets

HPC Project needs tools for its hardware accelerators (Wild Nodes from Wild Systems) and to parallelize, port & optimize customer applications

- 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...)
- Not reinventing the wheel (no NIH syndrome)

=> Funding an initiative to industrialize Open Source tools

Par4All is fully Open-Source (mix of MIT/GPL license)

According to Keshav Pingali, we're wrong at raising automatic parallelization from low-level code. But we provide generality across different tools, each with its own high-abstraction.
Par4All overview

- PIPS is the first project to enter the Par4All initiative
- Presented at Impact 2011: *PIPS Is not (just) Polyhedral Software*
Demo

- Example: mandelbrot written in Scilab
- Converted to C using COLD, an in-house (commercial) scilab-to-C compiler
- The C code is processed by Par4All to target multi-core or GPU
- PIPS is inter-procedural and thus needs all the source code, we need to provide stubs for the Scilab runtime
Focus on array regions analyses

- Starting with Béatrice Creusillet thesis (1996)
- Find out what part of an array is read or written
- Approximation: may/must/exact
- Set of linear relations

Applications:
- Parallelization
- Array privatization
- Scalarization
- Statement isolation
- Memory footprint reduction using tiling
int triangular(int m, int n, double a[n][m]) {
    int h = n/2;

    for(int i = 0; i < h; i += 1) {
        for(int j = i; j < m-i; j += 1) {
            a[i][j] = f();
        }
    }
}
IN/OUT Regions

PIPS includes inter-procedural IN and OUT regions

- **IN** regions include part of the array read by a statement, for which the value was produced earlier in the program.

```c
int in_regions(int n, double a[n], double b[n], double c[n]) {
    // <a[PHI1]-OUT-EXACT-{0<=PHI1, PHI1+1<=n}>
    for(int i=0; i<n; i++) {
        a[i] = init();
        b[i] = init();
    }

    // <a[PHI1]-IN-EXACT-{0<=PHI1, PHI1+1<=n}>
    for(int i=0; i<n; i++) {
        b[i] = a[i]+1;
        c[i] = f(a[i], b[i]);
    }
}
```

No in region on `b`

Overwrite 1st `b` assignment
IN/OUT Regions

PIPS includes inter-procedural IN and OUT regions

- **OUT** regions include part of the array produced by a statement and that will be used later in the program

```c
int in_regions(int n, double a[n], double b[n], double c[n]) {
    // <a[PHI1]-OUT-EXACT-{0<=PHI1, PHI1+1<=n}>
    for(int i=0; i<n; i++) {
        a[i] = init();
        b[i] = init();
    }
    // <a[PHI1]-IN-EXACT-{0<=PHI1, PHI1+1<=n}>
    for(int i=0; i<n; i++) {
        b[i] = a[i]+1;
        c[i] = f(a[i], b[i]);
    }
}
```

Nobody would write such code....

No in region on b means that a scalarization is possible
IN/OUT Regions

PIPS includes inter-procedural IN and OUT regions

- **OUT** regions include part of the array produced by a statement and that will be used later in the program

```c
int in_regions(int n, double a[n], double b[n], double c[n]) {
  // <a[PHI1]-OUT-EXACT-{0<=PHI1, PHI1+1<=n}>
  for(int i=0; i<n; i++) {
    a[i] = init();
    b[i] = init();
  }
  // <a[PHI1]-IN-EXACT-{0<=PHI1, PHI1+1<=n}>
  for(int i=0; i<n; i++) {
    b[i] = a[i]+1;
    c[i] = f(a[i], b[i]);
  }
}
```

Nobody would write such code....

… but what about automatically generated code from higher level description?

No out region on **b** means that a scalarization is possible
Application to host-accelerator communications

```c
void kernel(int n, double X[n][n]) {
    int i1, i2;

    for (i1 = 0; i1 < n/2; i1++) { // Sequential
        for(i2 = i1; i2 < n-i1; i2++) { // Parallel
            X[n - 2 - i1][i2] = X[n - 2 - i1][i2] - X[n - i1 - 3][i2];
        }
    }

    int main(int argc, char **argv) {
        if(argc!=2) {
            fprintf(stderr,"Size expected as first argument\n");
            exit(1);
        }

        int size = atoi(argv[1]); // Unsafe !
        double (*X)[size] = (double (*)[size])malloc(sizeof(double)*size*size);
        double (*A)[size] = (double (*)[size])malloc(sizeof(double)*size*size);
        double (*B)[size] = (double (*)[size])malloc(sizeof(double)*size*size);
        kernel(size,X,A,B);
    }
```
Application to host-accelerator communications

// \langle PHI1\langle PHI2\rangle-R-MAY-\{PHI2<=PHI1+2, n<=PHI1+PHI2+3, n<=2PHI1+4, PHI1+2<=n, 0<=PHI2, PHI2+1<=n, 2<=n\}\rangle
// \langle PHI1\langle PHI2\rangle-W-MAY-\{PHI2<=PHI1+1, n<=PHI1+PHI2+2, n<=2PHI1+2, PHI1+2<=n\}\rangle
for (i1 = 0; i1 < n/2; i1++) { // Sequential
// \langle PHI1\langle PHI2\rangle-R-EXACT-\{n<=PHI1+i1+3, PHI1+i1+2<=n, i1<=PHI2, PHI2+i1+1<=n\}\rangle
// \langle PHI1\langle PHI2\rangle-W-EXACT-\{PHI1+i1==n-2, i1<=PHI2, PHI2+i1+1<=n\}\rangle
for(i2 = i1; i2 < n-i1; i2++) { // Parallel
// \langle PHI1\langle PHI2\rangle-R-EXACT-\{PHI2==i2, n<=PHI1+i1+3, PHI1+i1+2<=n, i1<=PHI2, PHI2+i1+1<=n\}\rangle
// \langle PHI1\langle PHI2\rangle-W-EXACT-\{PHI1+i1==n-2, PHI2==i2, 0<=i1, i1<=i2\}\rangle
X[n - 2 - i1][i2] = X[n - 2 - i1][i2] - X[n - i1 - 3][i2];
}
}
Application to host-accelerator communications

// <X[PHI1][PHI2]-R-MAY-{PHI2<=PHI1+2, n<=PHI1+PHI2+3, n<=2PHI1+4, PHI1+2<=n, 0<=PHI2, PHI2+1<=n, 2<=n}>
// <X[PHI1][PHI2]-W-MAY-{PHI2<=PHI1+1, n<=PHI1+PHI2+2, n<=2PHI1+2, PHI1+2<=n}>
for (i1 = 0; i1 < n/2; i1++) { // Sequential
  // <X[PHI1][PHI2]-R-EXACT-{n<=PHI1+i1+3, PHI1+i1+2<=n, i1<=PHI2, PHI2+i1+1<=n}>
  // <X[PHI1][PHI2]-W-EXACT-{PHI1+i1==n-2, PHI2+i1+1<=n}>
  for(i2 = i1; i2 < n-i1; i2++) { // Parallel
    // <X[PHI1][PHI2]-R-EXACT-{PHI2==i2, n<=PHI1+i1+3, PHI1+i1+2<=n, i1<=PHI2, PHI2+i1+1<=n}>
    // <X[PHI1][PHI2]-W-EXACT-{PHI1+i1==n-2, PHI2==i2, 0<=i1, i1<=i2}>
    X[n - 2 - i1][i2] = X[n - 2 - i1][i2] - X[n - i1 - 3][i2];
  }
}
Application to host-accelerator communications

```c
for (i1 = 0; i1 < n/2; i1++) { // Sequential
    for (i2 = i1; i2 < n-i1; i2++) { // Parallel
        X[n - 2 - i1][i2] = X[n - 2 - i1][i2] - X[n - i1 - 3][i2];
    }
}
```

Read
Read and Written
Written on previous iterations
Application to host-accelerator communications

// <X[PHI1][PHI2]-R-MAY-{PHI2<=PHI1+2, n<=PHI1+PHI2+3, n<=2PHI1+4, PHI1+2<=n, 0<=PHI2, PHI2+1<=n, 2<=n}>
// <X[PHI1][PHI2]-W-MAY-{PHI2<=PHI1+1, n<=PHI1+PHI2+2, n<=2PHI1+2, PHI1+2<=n}>
for (i1 = 0; i1 < n/2; i1++) { // Sequential
// <X[PHI1][PHI2]-R-EXACT-{n<=PHI1+i1+3, PHI1+i1+2<=n, i1<=PHI2, PHI2+i1+1<=n}>
// <X[PHI1][PHI2]-W-EXACT-{PHI1+i1==n-2, i1<=PHI2, PHI2+i1+1<=n}>
    for(i2 = i1; i2 < n-i1; i2++) { // Parallel
// <X[PHI1][PHI2]-R-EXACT-{PHI2==i2, n<=PHI1+i1+3, PHI1+i1+2<=n, i1<=PHI2, PHI2+i1+1<=n}>
// <X[PHI1][PHI2]-W-EXACT-{PHI1+i1==n-2, PHI2==i2, 0<=i1, i1<=i2}>
    X[n - 2 - i1][i2] = X[n - 2 - i1][i2] - X[n - i1 - 3][i2];
    }
}
Application to host-accelerator communications

for (i1 = 0; i1 < n/2; i1++) { // Sequential
    for (i2 = i1; i2 < n-i1; i2++) { // Parallel
        X[n - 2 - i1][i2] = X[n - 2 - i1][i2] - X[n - i1 - 3][i2];
    }
}
Application to host-accelerator communications

```c
for (i1 = 0; i1 < n/2; i1++) { // Sequential
    // Allocate all the array on the accelerator
    double (*accel_X)[2][-2*i1+n];
    P4A_accel_malloc((void **) &accel_X, sizeof(double)*i1*2));
    Copy_to_accel_2d(sizeof(double), n, n, 2, -2*i1+n, -i1+n-3, i1, &X[0][0], *accel_X);
    for(i2 = 0; i2 < n-i1-i1; i2++) { // Parallel (has been skewed to start from 0)
        accel_X[1][i2] = accel_X[1][i2] - accel_X[0][i2];
    }
    Copy_from_accel_2d(
        n, n, // host size
        1, -2*i1+n, // transfer
        -i1+n-2, i1, // offset
        &X[0][0],
        &accel_X[1][0]);
    Accel_free(accel_X);
}
```

Rectangular hull
Exact data read by the kernel

Data written by the kernel
Read
Written on previous iterations
Read and Written
Data transferred on current iteration
Application to host-accelerator communications

for (i1 = 0; i1 < n/2; i1++) { // Sequential
    // Allocate all the array on the accelerator
    double (*accel_X)[2][-2*i1+n];
    P4A_accel_malloc((void **) &accel_X, sizeof(double)*i1*2));
    Copy_to_accel_2d(sizeof(double), n, n, 2, -2*i1+n, -i1+n-3, i1, &X[0][0], *accel_X);
    for (i2 = 0; i2 < n-i1-i1; i2++) { // Parallel (has been skewed to start from 0)
        accel_X[1][i2] = accel_X[1][i2] - accel_X[0][i2];
    }
    Copy_from_accel_2d(
        sizeof(double),
        n, n, // host size
        1, -2*i1+n, // transfer
        -i1+n-2, i1, // offset
        &X[0][0],
        &accel_X[1][0]);
    Accel_free(accel_X);
}
Application to host-accelerator communications

Can we avoid redundant transfers? Try a subtraction:

\[ \begin{align*}
&<X[\text{PHI1}][\text{PHI2}]-\{n<=\text{PHI1}+i1+3, \text{PHI1}+i1+2\leq n, \\
i1<=\text{PHI2}, \text{PHI2}+i1+1\leq n}\rangle \\
- \\
&<X[\text{PHI1}][\text{PHI2}]-\{n<=\text{PHI1}+(i1-1)+3, \text{PHI1}+(i1-1)+2\leq n, \\
(i1-1)<=\text{PHI2}, \text{PHI2}+(i1-1)+1\leq n}\rangle \\
= \\
<X[\text{PHI1}][\text{PHI2}]-\{n==\text{PHI1}+i1+3, \\
i1<=\text{PHI2}, \text{PHI2}+i1+1\leq n}\rangle
\end{align*} \]

From Alias, Darte, and Plesco, Impact 2012:

\[ \text{In}(l1) \setminus \text{Out}(i1 < l1) \subseteq \text{Load}(i1 \leq l1) \]
\[ \text{Out}(i1 < l1) \cap \text{Load}(l1) = \emptyset \]
Application to host-accelerator communications

```c
// <X[PHI1][PHI2]-R-MAY-{PHI2<=PHI1+2, n<=PHI1+PHI2+3, n<=2PHI1+4, 
  PHI1+2<=n, 0<=PHI2, PHI2+1<=n, 2<=n}>
// <X[PHI1][PHI2]-W-MAY-{PHI2<=PHI1+1, n<=PHI1+PHI2+2, n<=2PHI1+2, PHI1+2<=n}>

double (*accel_X)[n-2-(n/2-1)+1][n-1+1];
P4A_accel_malloc((void **) &accel_X, sizeof(double)*(n-2-(n/2-1)+1)*(n-1+1));
// Data for first iteration
Copy_to_accel_2d(sizeof(double), n, n, 1, n, n-3, 0, &X[0][0], &accel_X[n-2-(n/2-1)+1][0]);
for (i1 = 0; i1 < n/2; i1++) { // Sequential
  Copy_to_accel_2d(sizeof(double), n, n, 1,-2*i1+n,-i1+n-3-2-(n/2-1)+1, i1, &X[0][0],*accel_X);
  for(i2 = 0; i2 < n-i1-i1; i2++) // Parallel
    X[n - 2 - i1-2-(n/2-1)+1][i2] = X[n - 2 - i1-2-(n/2-1)+1][i2] - X[n - i1 – 3-2-(n/2-1)+1][i2];
  Copy_from_accel_2d(
    sizeof(double),
    n, n, // host size
    1, -2*i1+n, // transfer
    -i1+n-2, i1, // offset
    &X[0][0],
    &accel_X[1][0]);
}
Accel_free(accel_X);
```
Par4All future

Source code (with directives?)

PIPS
Transformations & Analyses

kernels

Post-processor, optimizer, ...

nvcc-like

host code

Post-processor, optimizer, ...

host compiler

Final Binary

feature extractor

polyhedral optimizer

OpenSCoP

Par4All Runtime

Other Runtimes
Par4All future

Source code (with directives?)

PIPS
Transformations & Analyses

Your ?
feature extractor

Your ?
polyhedral optimizer

OpenSCoP

Your ?
Post-processor, optimizer, ...

host code
Par4All Runtime

Other Runtimes Including yours ?

Your ?
Post-processor, optimizer, ...

kernels
nvcc-like

host compiler

Final Binary

Including yours ?

Other Runtimes

Including yours ?