SPIRE: A Methodology for Sequential to Parallel Intermediate Representation Extension

Dounia Khaldi Pierre Jouvelot François Irigoin Corinne Ancourt

CRI, Mathématiques et systèmes
MINES ParisTech
35 rue Saint-Honoré, 77300 Fontainebleau, France

HiPEAC Computing Systems Week, Paris, France
May 03, 2013

Context and Motivation

- Cilk, UPC, X10, Habanero-Java, Chapel, OpenMP, MPI, OpenCL...
- Parallelism handling in compilers
- A Parallel Intermediate Representation
  - Trade-off between expressibility and conciseness of representation
  - Generic (language-neutral) and simplicity
- Huge compiler platforms
  - GCC (more than 7 million lines of code)
  - LLVM (more than 1 million lines of code)
  - PIPS (600 000 lines of code)

Proposal

SPIRE: Sequential to Parallel Intermediate Representation Extension methodology

Rich Related Work !

	Sequential IR	Parallel IR	Parallelism
AST	GIMPLE [Merrill, 2003]	GCC (GOMP) [Novillo, 2006]	Additional built-ins (`__sync_fetch_and_add`)
	Habanero-Java [Cavé et al., 2011] AST IR	HPIR [Zhao and Sarkar, 2011]	New nodes: `AsyncRegionEntry`, `AsyncRegionExit` `FinishRegionEntry`, `FinishRegionExit`
	—	InsPIRe [Insieme,]	Built-ins
	—	PLASMA [Pai et al., 2010]	Vector operators: `add`, `reduce` and `par`
Graph	Program Dependence Graph [Ferrante et al., 1987]	Parallel Program Graph [Sarkar and Simons, 1993]	mgoto control and synchronization edges
	—	Hierarchical Task Graph [Girkar and Polychronopoulos, 1992]	New nodes for tasks
	—	Stream Graph [Choi et al., 2009]	Nodes for streams
	LLVM IR [Team, 2010]	LLVM PIR	Metadata `llvm.loop.parallel`

Outline

Design Approach

"parallelism or concurrency are operational concepts that refer not to the program, but to its execution." [Dijkstra, 1977]

Language	Execution	Synchronization				Memory	
	Parallelism	Task creation	Task join	Point-to-point	Atomic section	Model	Data distribution
Cilk (MIT)	—	spawn	sync	—	cilk_lock	Shared	—
Chapel (Cray)	forall coforall cobegin	begin	—	sync	sync atomic	PGAS (Locales)	(on)
X10(IBM), Habanero-Java(Rice)	foreach	async future	finish	next force get	atomic isolated	PGAS (Places)	(at)
OpenMP	omp for omp sections	omp task omp section	omp taskwait omp barrier	—	omp critical omp atomic	Shared	private, shared...
OpenCL	EnqueueND-RangeKernel	EnqueueTask	Finish EnqueueBarrier	events	atom_add, ...	Distributed	ReadBuffer WriteBuffer
MPI	MPI_Init	MPI_spawn	MPI_Finalize MPI_Barrier	—	—	Distributed	MPI_Send MPI_Recv
SPIRE	**sequential**, **parallel**	**spawn**	**barrier**	**signal**, **wait**	**atomic**	Shared, Distributed	**send**, **recv**

PIPS (Sequential) IR in Newgen

```
instruction   = call + forloop + sequence;

statement     = instruction x declarations:entity*;

entity        = name:string x type x initial:value;

forloop       = index:entity x
                lower:expression x upper:expression x
                step:expression x body:statement;

sequence      = statements:statement*;
```

SPIRE: Execution

```
execution      = sequential:unit + parallel:unit;
```

- Add `execution` to control constructs:
  - `loop`
  - `sequence`

```
forall I in 1..n do
    t[i] = 0;
```

```
forloop(I,1,n,1,
        t[i] = 0,
        parallel)
```

`forall` in Chapel, and its SPIRE core language representation

SPIRE: Synchronization

```
synchronization = none:unit +
                  spawn:entity + barrier:unit +
                  single:bool + atomic:reference;
```

- Add `synchronization` to `statement`

```
mode = OUT_OF_ORDER_EXEC_MODE_ENABLE;
c = clCreateCommandQueue(context,
              device_id,mode,&err);
clEnqueueTask(c, k_A, 0, NULL, NULL);
clEnqueueTask(c, k_B, 0, NULL, NULL);
clEnqueueBarrier(c);
clEnqueueTask(c, k_C, 0, NULL, NULL);
```

```
barrier(
 spawn(zero,k_A(...));
 spawn(one,k_B(...))
);
spawn(zero,k_C(...))
```

OpenCL example illustrating spawn and barrier statements, and its SPIRE core language representation

SPIRE: Point-to-Point Synchronization (Event API)

```
event newEvent(int i);
void freeEvent(event e);
void signal(event e);
void wait(event e);
```

- Add `event` as a new type

```
finish{
 phaser ph=new phaser();
 for(j = 1;j <= n;j++){
     async phased(
         ph<SIG_WAIT>){
            S; next; S′;
     }
 }
}
```

```
barrier(
  ph=newEvent(-(n-1));
  forloop(j, 1, n, 1,
          spawn(j, S;
                signal(ph);
                wait(ph);
                signal(ph);
                S′),
          parallel);
  freeEvent(ph)
)
```

A phaser in Habanero-Java, and its SPIRE core language representation

Sequential Core Language

$S \in$ Stmt::= nop | I=E | $S_1;S_2$ | loop(E,S)

$S \in$ **SPIRE**(Stmt)::=
nop | I=E | $S_1;S_2$ | loop(E,S) |
spawn(I,S) |
barrier(S) |
wait(I) | **signal**(I) |
send(I,I′) | **recv**(I,I′)

$m \in \text{Memory} = \text{Ide} \to \text{Value}$
$\kappa \in \text{Configuration} = \text{Memory} \times \text{Stmt}$
$\zeta \in \text{Exp} \to \text{Memory} \to \text{Value}$

$$\frac{v = \zeta(E)m}{(m, I = E) \to (m[I \to v], \text{nop})}$$

SPIRE as a Language Transformer

$\kappa \in \text{Configuration} = \text{Memory} \times \texttt{Stmt}$
$\pi \in \text{State} = \text{Proc} \to \text{Configuration} \times \text{Procs}$
$\text{i} \in \text{Proc} = \text{N}$
$\text{c} \in \text{Procs} = \mathcal{P}(\text{Proc})$
$\text{dom}(\pi) = \{\text{i} \in \text{Proc} / \pi(\text{i}) \text{ is defined}\}$
$\pi[\text{i} \to (\kappa, \text{c})]$ the state π extended at i with (κ, c)

$$\frac{\kappa \to \kappa'}{\pi[\text{i} \to (\kappa, \text{c})] \hookrightarrow \pi[\text{i} \to (\kappa', \text{c})]}$$

$$\frac{\text{n} = \zeta(\texttt{I})\text{m}}{\begin{array}{l}\pi[\text{i} \to ((\text{m}, \texttt{spawn(I,S)}), \text{c})] \hookrightarrow \\ \qquad \pi[\text{i} \to ((\text{m}, \texttt{nop}), \text{c} \cup \{\text{n}\})] \\ \qquad\quad [\text{n} \to ((\text{m}, \texttt{S}), \emptyset)]\end{array}}$$

Validation: LLVM

```
function    = blocks:block*;
block       = label:entity x phi_nodes:phi_node* x
              instructions:instruction* x terminator;
phi_node    = call;
instruction = call;
terminator  = conditional_branch + unconditional_branch +
              return;
conditional_branch = value:entity x label_true:entity x
                     label_false:entity;
unconditional_branch = label:entity;
return     = value:entity;
```

Validation: LLVM

```
function    = blocks:block*;
block       = label:entity x phi_nodes:phi_node* x
              instructions:instruction* x terminator;
phi_node    = call;
instruction = call;
terminator  = conditional_branch + unconditional_branch +
              return;
conditional_branch = value:entity x label_true:entity x
                     label_false:entity;
unconditional_branch = label:entity;
return     = value:entity;
```

```
sum = 42;
for(i=0; i<10; i++){
  sum = sum + 2;
}
```

```
entry:
 br label %bb1
bb:                ; preds = %bb1
 %0 = add nsw i32 %sum.0, 2
 %1 = add nsw i32 %i.0, 1
 br label %bb1
bb1:               ; preds = %bb, %entry
 %sum.0 = phi i32 [42,%entry],[%0,%bb]
 %i.0 = phi i32 [0,%entry],[%1,%bb]
 %2 = icmp sle i32 %i.0, 10
 br i1 %2, label %bb, label %bb2
```

Validation: SPIRE (LLVM)

```
function    = blocks:block*;
block       = label:entity x phi_nodes:phi_node* x
              instructions:instruction* x terminator;
phi_node    = call;
instruction = call;
terminator  = conditional_branch + unconditional_branch +
              return;
conditional_branch = value:entity x label_true:entity x
                     label_false:entity;
unconditional_branch = label:entity;
return     = value:entity;
```

Validation: SPIRE (LLVM)

```
function    = blocks:block*;
block       = label:entity x phi_nodes:phi_node* x
              instructions:instruction* x terminator;
phi_node    = call;
instruction = call;
terminator  = conditional_branch + unconditional_branch +
              return;
conditional_branch = value:entity x label_true:entity x
                     label_false:entity;
unconditional_branch = label:entity;
return     = value:entity;
```

```
function    ⇝ function x execution;
block       ⇝ block x execution;
instruction ⇝ instruction x synchronization;
Intrinsic Functions: send, recv, signal, wait
```

Conclusion

- SPIRE methodology as an IR transformer:
  - 10 concepts collected in three groups
    execution, synchronization, data distribution
  - SPIRE (PIPS), SPIRE (LLVM)
  - Operational semantics for SPIRE [Khaldi et al., 2012b]
  - Trade-off between expressibility and conciseness of representation
- Applications:
  - Implementation of SPIRE (PIPS)
  - Implementation of a new BDSC-based task parallelization algorithm [Khaldi et al., 2012a]
  - Generation of OpenMP and MPI code from the same parallel IR

Future Work

- Transformations for parallel languages encoded in SPIRE
- Representation of the PGAS memory model
- Addition of programming features such as exceptions and speculation

References

Cavé, V., Zhao, J., and Sarkar, V. (2011).
Habanero-Java: the New Adventures of Old X10.
In 9th International Conference on the Principles and Practice of Programming in Java (PPPJ).

Choi, Y., Lin, Y., Chong, N., Mahlke, S., and Mudge, T. (2009).
Stream Compilation for Real-Time Embedded Multicore Systems.
In Proceedings of the 7th annual IEEE/ACM International Symposium on Code Generation and Optimization, CGO '09, pages 210–220, Washington, DC, USA.

Dijkstra, E. W. (1977).
Re: "Formal Derivation of Strongly Correct Parallel Programs" by Axel van Lamsweerde and M.Sintzoff.
circulated privately.

Ferrante, J., Ottenstein, K. J., and Warren, J. D. (1987).
The Program Dependence Graph And Its Use In Optimization.
ACM Trans. Program. Lang. Syst., 9(3):319–349.

Girkar, M. and Polychronopoulos, C. D. (1992).
Automatic Extraction of Functional Parallelism from Ordinary Programs.
IEEE Trans. Parallel Distrib. Syst., 3:166–178.

Insieme.
Insieme - an Optimization System for OpenMP, MPI and OpenCL Programs.
http://www.dps.uibk.ac.at/insieme/architecture.html.

Khaldi, D., Jouvelot, P., and Ancourt, C. (2012a).
Parallelizing with BDSC, a Resource-Constrained Scheduling Algorithm for Shared and Distributed Memory Systems.
Technical Report CRI/A-499 (Submitted to Parallel Computing), MINES ParisTech.

Khaldi, D., Jouvelot, P., Ancourt, C., and Irigoin, F. (2012b).
SPIRE: A Sequential to Parallel Intermediate Representation Extension.
Technical Report CRI/A-487 (Submitted to CGO'13), MINES ParisTech.

SPIRE: A Methodology for Sequential to Parallel Intermediate Representation Extension

Dounia Khaldi Pierre Jouvelot François Irigoin Corinne Ancourt

CRI, Mathématiques et systèmes
MINES ParisTech
35 rue Saint-Honoré, 77300 Fontainebleau, France

HiPEAC Computing Systems Week, Paris, France
May 03, 2013

SPIRE: Data Distribution

```
void send(int dest, entity buf);
void recv(int source, entity buf);
```

```
MPI_Init(&argc, &argv[]);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
if (rank == 0)
  MPI_Recv(sum,sizeof(sum),MPI_FLOAT,
           1,1,MPI_COMM_WORLD,&stat);
else if(rank == 1){
  sum = 42;
  MPI_Send(sum,sizeof(sum),MPI_FLOAT,
           0,1,MPI_COMM_WORLD);
}
MPI_Finalize();
```

```
forloop(rank,0,size,1,
        test(rank==0,
             recv(one,sum),
             test(rank==1,
                  sum=42;
                  send(zero,sum),
                  nop)
             ),
        parallel)
```

SPIRE: Data Distribution

```
void send(int dest, entity buf);
void recv(int source, entity buf);
```

```
MPI_Init(&argc, &argv[]);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
if (rank == 0)
  MPI_Recv(sum,sizeof(sum),MPI_FLOAT,
           1,1,MPI_COMM_WORLD,&stat);
else if(rank == 1){
  sum = 42;
  MPI_Send(sum,sizeof(sum),MPI_FLOAT,
           0,1,MPI_COMM_WORLD);
}
MPI_Finalize();
```

```
forloop(rank,0,size,1,
        test(rank==0,
             recv(one,sum),
             test(rank==1,
                  sum=42;
                  send(zero,sum),
                  nop)
             ),
        parallel)
```

SPIRE: Data Distribution

- Non-blocking send ≡

```
spawn(new,send(zero,sum))
```

- Non-blocking receive ≡

```
finish_recv = false;
spawn(new,recv(one,sum);
          atomic(finish_recv=true))
```

- Broadcast ≡

```
test(rank==0,
     forloop(rank,1,size,1,
             send(rank,sum),
             parallel),
     recv(zero,sum))
```