
Productivity via Automatic Code Generation for PGAS Platforms with the R-Stream compiler

Benoît Meister, Allen Leung, Nicolas Vasilache,
David Wohlford, Cédric Bastoul, Richard Lethin

Reservoir Labs, Inc.

{meister,leunga,vasilache,wohlford,bastoul,lethin}@reservoir.com

Workshop on Asynchrony in the PGAS Programming Model

Large scale: programmability and power efficiency remain major issues [Exa08]

- Programmability improved by:
 - High-level programming language abstractions, e.g., HPF, Chapel
 - APIs for abstracting Hardware, e.g. PGAS
 - Adoption: extending familiar languages [Lang]

- Performance \cap power efficiency: communications are the bottleneck
 - Locality of reference
 - Exploiting hardware mechanism: bulk communications
 - Communication-computation overlap
 - Explicit management of *asynchronous* communications [NHB09,CBI07]

Large scale: programmability and power efficiency remain major issues [Exa08]

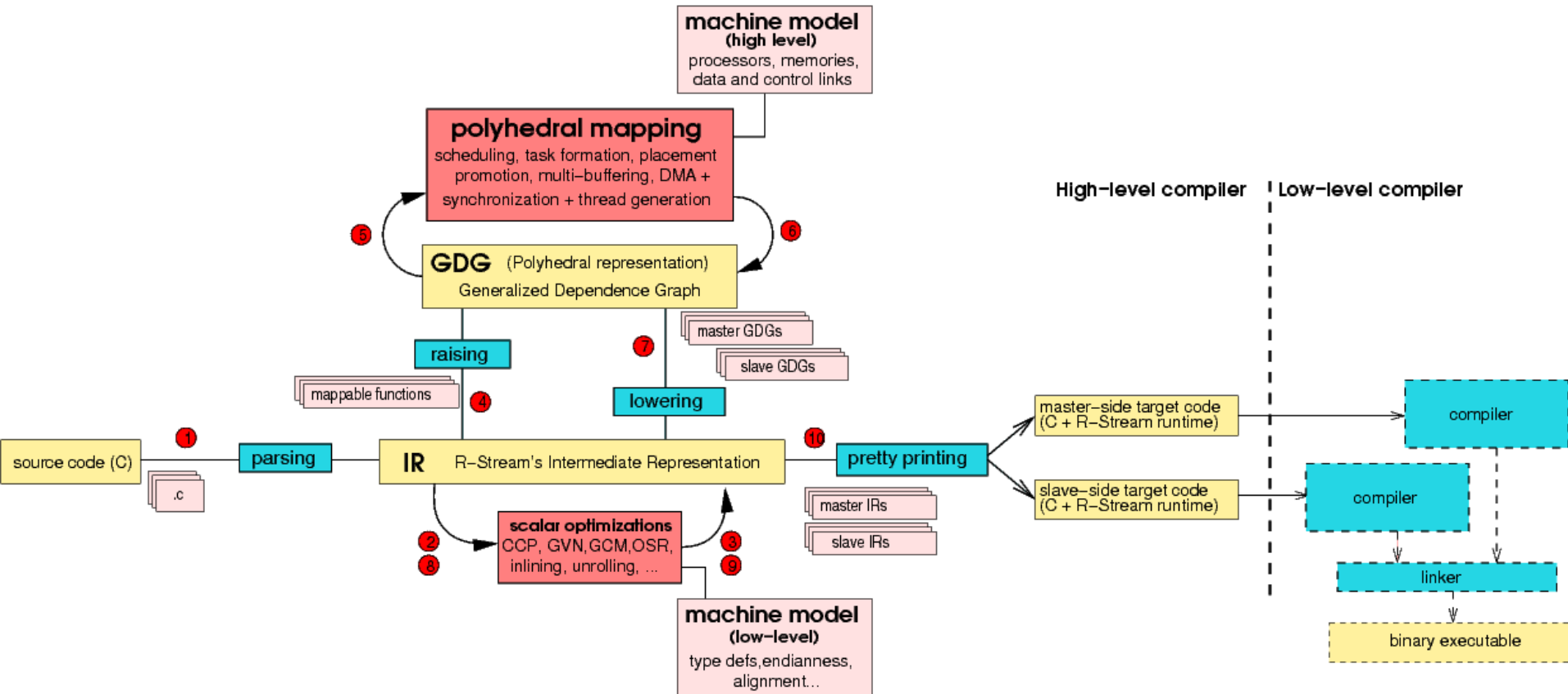
- Programmability improved by:
 - High-level programming language abstractions, e.g., HPF, Chapel
 - APIs for abstracting Hardware, e.g. PGAS
 - Adoption: extending familiar languages [Lang]

R-Stream: C compiler using High-level abstractions to target Hardware
mechanism abstraction APIs

- Performance \cap power efficiency: communications are the bottleneck
 - Locality of reference
 - Exploiting hardware mechanism: bulk communications
 - Communication-computation overlap
 - Explicit management of *asynchronous* communications [NHB09,CBI07]

Multi-buffering: a way to enable communication-computation overlap

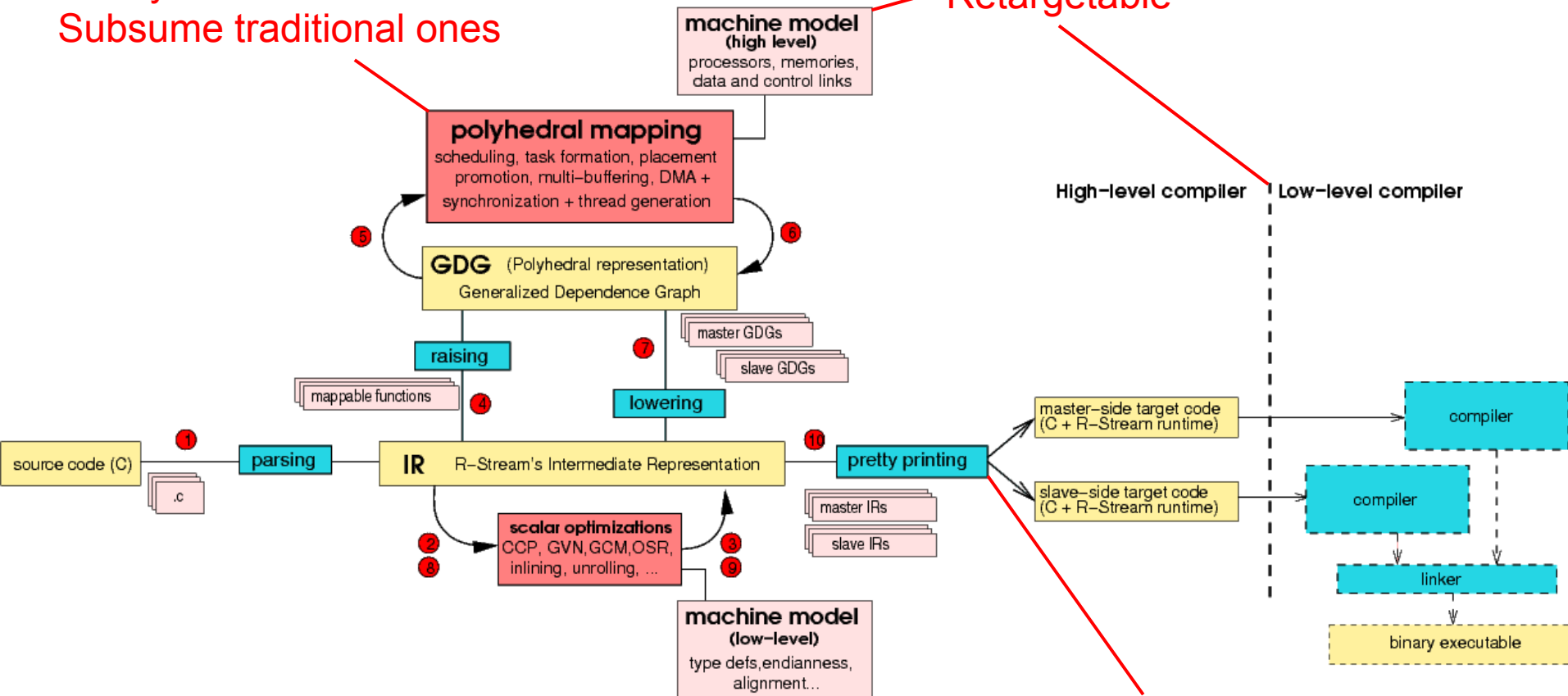
R-Stream: source-to-source compiler based on polyhedral abstractions



R-Stream: source-to-source compiler based on polyhedral abstractions

Not syntax-based transformations
Subsume traditional ones

Retargetable



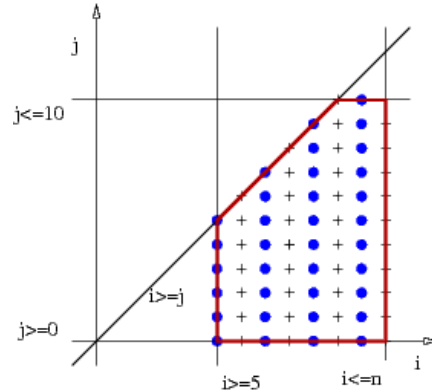
Can target many languages and abstraction APIs

Polyhedral representation

```

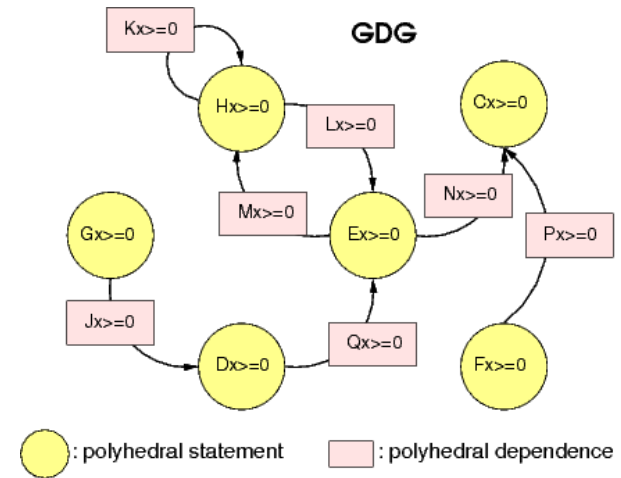
n = f();
for (i=5; i<= n; i+=2) {
  A[i] = A[i]/B[i];
  for (j=0; j<=i; j++) {
    if (j<=10) {
      ... A[i+2j+n][i+3]...
    }
  }
}

```



$$\{i, j \in \mathbb{Z}^2 \mid \exists k \in \mathbb{Z}, 5 \leq i \leq n; 0 \leq j \leq i; j \leq 10; i = 2k + 1\}$$

$$\begin{pmatrix} A_0 \\ A_1 \\ 1 \end{pmatrix} = \begin{bmatrix} 1 & 2 & 1 & 0 \\ 1 & 0 & 0 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \\ n \\ 1 \end{bmatrix}$$



Affine and non-affine transformations
Order and place of operations and data

Base model: affine static control programs.

Exact representation

Extended model(s): broader application domain (data-dependent, infinite, less regular loops)

Approximate (but correct) representation

Mapping process for Cell

- Expose coarse-grain parallelism and locality
- Regroup operations into atomic tasks and place on processors
- Optimize local memory layout
- Generate asynchronous bulk communications
- **Set up multi-buffering**
- Generate synchronizations
- Partition mapped function into master (PowerPC) and slave (SPE) code and generate control flow transfer between them
- Processor-specific optimizations

A multi-buffering approach for the Cell processor

- Duplicate working data sets into B “buffers”
 - Initiate receiving input data for task T-A while executing task T
 - Complete sending output data for task T+C
 - Rotate buffer roles between consecutive tasks
-
- Cell is a good model for APGAS
 - Distributed, explicitly managed memory
 - Asynchronous communications
 - Common, distant memory and local (close) memory

Algorithm walkthrough: 1024x1024 matrix multiply

```
matmult_PE(float ** A,B,C, int PE_ID) {
  float C_l[64][512], A_l[64][16], B_l[16][512];
  for (i = 0, i<=1; i++) {
    for(j=0; j<=1; j++) {
      init(C_l[k][l], 64, 512);
      for (k=0; k<=63; k++) {
        DMA_get(&B[16*k][512*j], //src@
               &B_l[0][0],      //dst@
               2048,             //size
               4096, 2048,      //strides
               16,               //count
               0);              // tag
        DMA_get(&A[512*i+64*PE_ID],
               &A_l[0][0],
               64, 4096, 64, 64, 0);
        DMA_wait(0);
        update(C_l, A_l, B_l, 64, 512, 16);
      }
      DMA_put(&C_l[0][0],
             &C[512*i+64*PE_ID][256*j],
             2048, 2048, 4096, 64, 1);
      DMA_wait(1);
    }
  }
}
```

1. Select pipeline-prone loop level L_p
 - scans series of comm and computations
 - D: transferred data set (here: A,B)
2. Select “buffer” loop L_b
 - short ≥ 0 dependence distance on D
 - sinkable down to L_p
 - long enough trip count

Domain (DMA_gets and update):

$$\{0 \leq i \leq 1; 0 \leq j \leq 1; 0 \leq k \leq 63\}$$



■ : DMA_get ■ : update ■ : both

Algorithm walkthrough

```
matmult_PE(float ** A,B,C, int PE_ID) {
  float C_l[64][256], A_l_buf[2][64][16],
        B_l_buf[2][16][256];
  float (*A_l)[16], (*A_l_1)[16];
  float (*B_l)[256], (*B_l_1)[256];
  A_l = A_l_buf[0]; A_l_1 = A_l_buf[1];
  B_l = B_l_buf[0]; B_l_1 = B_l_buf[1];
  for (i = 0, i<=1; i++) {
    for(j=0; j<=3; j++) {
      init(C_l[k][l], 64, 512);
      for (k=0; k<=63; k++) {
        DMA_get(&B[16*k][256*j], //src@
               &B_l[0][0],      //dst@
               1024,             //size
               4096, 1024,      //strides
               16,               //count
               0);               // tag
        DMA_get(&A[512*i+64*PE_ID],
               &A_l[0][0],
               64, 4096, 64, 64, 0);
        DMA_wait(0);
        update(C_l, A_l, B_l, 64, 512, 16);
      }
      DMA_put(&C_l[0][0],
             &C[256*i+64*PE_ID][256*j],
             1024, 1024, 4096, 64, 1);
      DMA_wait(1);
    }
  }
}
```

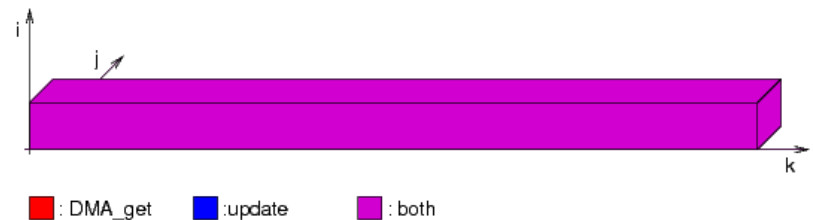
3. Produce B (here 2) versions of data in D

4. If limited memory, adapt task size

5. Sink Lb below Lp (here: identity)

Domain (DMA_gets and update):

$$\{0 \leq i \leq 1; 0 \leq j \leq 1; 0 \leq k \leq 63\}$$



Algorithm walkthrough

```

matmult_PE(float ** A,B,C, int PE_ID) {
  float C_l[64][256], A_l_buf[2][64][16],
        B_l_buf[2][16][256];
  A_l = A_l_buf[0]; A_l_1 = A_l_buf[1];
  B_l = B_l_buf[0]; B_l_1 = B_l_buf[1];
  for (i = 0, i<=1; i++) {
    for(j=0; j<=3; j++) {
      init(C_l[k][l], 64, 512);
      for (k=-1; k<=63; k++) {
        if (k<=62) {
          DMA_get(&B[16*(k+1)][256*j], //src@
                 &B_l_1[0][0],      //dst@
                 1024,                //size
                 4096, 1024,          //strides
                 16,                  //count
                 0);                  // tag
          DMA_get(&A[512*i+64*PE_ID][16*(k+1)],
                 &A_l_1[0][0],
                 64, 4096, 64, 64, 0);
          if (k>=0) {
            rotate(A_l,A_l_1); rotate(B_l,B_l_1);
            DMA_wait(0);
            update(C_l, A_l, B_l, 64, 512, 16);
          }
        }
      }
      DMA_put(&C_l[0][0],
             &C[256*i+64*PE_ID][256*j],
             1024, 1024, 4096, 64, 1);
      DMA_wait(1);
    }
  }
}

```

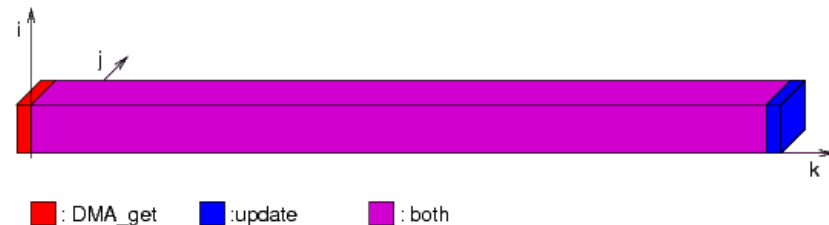
6. Re-schedule the “initiate receive” operations on D by $-A$ along L_b
 \rightarrow here: shift DMA_gets by $(0, 0, -1)^T$
7. Re-schedule the “complete send” operations on D by C along L_b (none here)
8. Insert buffer rotation operations at each iteration of L_b

Domain (DMA_wait and update):

$$\{0 \leq i \leq 1; 0 \leq j \leq 1; 0 \leq k \leq 63\}$$

Domain (DMA_gets):

$$\{0 \leq i \leq 1; 0 \leq j \leq 1; -1 \leq k \leq 62\}$$



Code generation is “free” in R-Stream

```
matmult_PE(float ** A,B,C, int PE_ID) {
  float C_l[64][256], A_l_buf[2][64][16],
        B_l_buf[2][16][256];
  A_l = A_l_buf[0]; A_l_1 = A_l_buf[1];
  B_l = B_l_buf[0]; B_l_1 = B_l_buf[1];
  for (i = 0, i<=1; i++) {
    for(j=0; j<=3; j++) {
      init(C_l[k][l], 64, 512);
      for (k=-1; k<=63; k++) {
        if (k<=62) {
          DMA_get(&B[16*(k+1)][256*j], //src@
                 &B_l_1[0][0],      //dst@
                 1024,                //size
                 4096, 1024,          //strides
                 16,                  //count
                 0);                  // tag
          DMA_get(&A[512*i+64*PE_ID][16*(k+1)],
                 &A_l_1[0][0],
                 64, 4096, 64, 64, 0);
          if (k>=0) {
            rotate(A_l,A_l_1); rotate(B_l,B_l_1);
            DMA_wait(0);
            update(C_l, A_l, B_l, 64, 512, 16);
          }
        }
      }
      DMA_put(&C_l[0][0],
             &C[256*i+64*PE_ID][256*j],
             1024, 1024, 4096, 64, 1);
      DMA_wait(1);
    }
  }
}
```

lowering
produces this
automatically

polyhedral mapper
works on this

Domain (DMA_wait and update):
 $\{0 \leq i \leq 1; 0 \leq j \leq 1; 0 \leq k \leq 63\}$
Domain (DMA_gets):
 $\{0 \leq i \leq 1; 0 \leq j \leq 1; -1 \leq k \leq 62\}$



Level of success

- On the example:
(avg. over 5 runs)

Version	Exec. Time	GFLOP/s
Original	0.0299±0.0003	71.9±0.8
Multi-buffered	0.0217±0.0006	98.8±2.8
Multi-buffered, no DMA	0.0216±0.0005	99.4±2.3

- Implementing optimization in R-Stream's mapper: simple
 - A series of polyhedral operations. C code *generated*.
 - Scheduler provides good “buffer loop” candidates
 - Task formation sets communication granularity
- Programmability of explicitly managed memories
 - User writes a clean implementation, follow rules and restrictions.
 - No communications, synchronizations, target-specific optimizations

Conclusions

- R-Stream approach addresses overlap issue on the Cell
 - Close enough to PGAS with asynchronous transfers
- Explicit data transfer management : *one of* the facets of R-Stream
 - All aspects of parallel execution
 - R-Stream targets many different execution models
- Potential uses for APGAS:
 - APGAS APIs as a target to R-Stream
 - Improving PGAS-based compilers

References

- [Exa08]** Peter M. Kogge, Shekhar Borkar, William W. Carlson, William J. Dally, Monty Denneau, Paul D. Franzon, Stephen W. Keckler, Dean Klein, Robert F. Lucas, Steve Scott, Allan E. Snively, Thomas L. Sterling, R. Stanley Williams, Katherine A. Yelick, William Harrod, Daniel P. Campbell, Kerry L. Hill, Jon C. Hiller, Sherman Karp, Mark A. Richards, and Alfred J. Scarpelli. *Exascale Study Group: Technology Challenges in Achieving Exascale Systems*. Technical report, DARPA, 2008.
- [NHB09]** R. Nishtala, P. Hargrove, D. Bonachea, K. Yelick. *Scaling Communication-Intensive Applications on BlueGene/P Using One-Sided Communication and Overlap*. 23rd International Parallel & Distributed Processing Symposium (IPDPS), 2009.
- [CBI07]** W. Chen, D. Bonachea, C. Iancu, K. Yelick. Automatic Nonblocking Communication for Partitioned Global Address Space Programs. International Conference on Supercomputing (ICS), 2007
- [Lang]**
- The HPF Forum. *High Performance Fortran language specification, v1.0*. May 1993.
 - K. A. Yelick, L. Semenzato, G. Pike, C. Miyamoto, B. Liblit, A. Krishnamurthy, P. N. Hilfinger, S. L. Graham, D. Gay, P. Colella, and A. Aiken. *Titanium: A High-Performance Java Dialect*. In *Concurrency: Practice and Experience*, Vol. 10, No. 11-13, 1998.
 - Daniel Chavarria-Miranda, Cristian Coarfa, Yuri Dotsenko and John Mellor-Crummey. *An Emerging, Portable Co-Array Fortran Compiler for High-Performance Computing*. Los Alamos Computer Science Institute 3th Annual Symposium (LACSI 2002), Santa Fe, New Mexico, October 2002.

References

- W. Carlson, J. Draper, D. Culler, K. Yelick, E. Brooks, and K. Warren. *Introduction to UPC and Language Specification*. CCS-TR-99-157, IDA Center for Computing Sciences, 1999. PDF available.
- David Callahan, Bradford L. Chamberlain, Hans P. Zima. *The Cascade High Productivity Language*. In 9th International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS 2004), pages 52-60. IEEE Computer Society, April 2004.
- Eric Allen, David Chase, Joe Hallett, Victor Luchangco, Jan-Willem Maessen, Sukyoung Ryu, Guy L. Steele Jr., Sam Tobin-Hochstadt, Joao Dias, Carl Eastlund, Christine Flood, Yossi Lev, Cheryl McCosh, Janus Dam Nielsen, Dan Smith. *The Fortress Language Specification Version 1.0*. Sun Microsystems technical report. March 2008.
- Kemal Ebcioglu, Vijay Saraswat, Vivek Sarkar. *X10: Programming for Hierarchical Parallelism and Non-Uniform Data Access*. 3rd International Workshop on Language Runtimes, Impact of Next Generation Processor Architectures on Virtual Machine Technologies co-located with ACM OOPSLA.