ACOPAL

Analyse et Compilation des langages de programmation parallèles

Analysis and Compilation of Parallel Programming Languages

Abstract

Parallel programming is increasingly important for embedded systems as well as scientific computing because of the hardware evolution toward heterogeneous multiprocessors. However developers of parallel applications are still stuck with three main, relatively primitive approaches: low-level multithreading, OpenMP annotations for shared-memory multiprocessors, and message passing in MPI for distributed platforms. No recent parallel language has emerged as a serious contender.
The current compiler technology for such programming models breaks down with every kind of parallel constructs, because concurrency in these low-level models is always expressed as procedure calls, with unknown side effects. Even recent parallel languages, such as PGAS ones, are first translated into C or C++ by converting parallel constructs into procedure calls. So many, if not all, optimization passes are disabled, and no theoretical foundation or common practice exist for the construction of optimizing compilers for such languages.
The ACOPAL project seeks to establish solid formal foundations for high- and low-level parallel internal representations (PIR, HPIR, LPIR) able to support 1) multiple parallel paradigms and languages, 2) analyses of both parallel and sequential codes, and 3) optimizing transformations legal for both parallel and sequential codes, and to demonstrate its usefulness by implementing such a PIR in at least one existing tool.

Parallel internal representations are key to the evolution of production compilers such as GCC or LLVM, and to the construction of static analyzers for parallel programs. The cooperation between ENS and MINES ParisTech will lead to a PSL center of excellence in static program analysis and compilation.

Research Program :

Context

When hitting the power wall around 2004, processor architects had to look for other ways to use the ever increasing number of transistors they could pack on one chip. The number of cores started to climb, leading to multicores and then manycore processors. Specialization was introduced, leading to Graphical Processing Units (GPUs) and a variety of hardware accelerators for dedicated applications (signal and media processing, cryptography).

In the meantime, the energy necessary to execute a task became a new optimization target beside execution times, in order to sustain the development of mobile and ubiquitous systems. New heterogeneous and hardware combinations with increasing degrees of parallelism and specialization appear at an accelerated pace.

But the best hardware system architectures will not succed if programming them for performance and safety is too difficult and unproductive. To address this issue the US government launched an ambitious research program to develop new parallel programming languages in cooperation with US manufacturers. As a result, two new languages, Chapel [8] developed by Cray and X10 [6] by IBM, are now available as well as more languages such as CAF [39] and OpenSHMEM [5]. In the same time frame, the success of GPUs for increasingly general-purpose applications lead manufacturers and users to develop a new language to control low-level hardware features, such as local memories or SIMD units, has well as higher-level ones, such as the number of cores: OpenCL was born [4]. Yet, programmers mostly interested in portability stuck to their favorite tools, the library-based MPI [3] for scientific programming and distributed machines, the annotation-based OpenMP [7] for shared-memory parallel processing, and combinations of both when dealing with clusters of shared-memory multiprocessors. The MPI and OpenMP, as well as C, C++ and Cilk [1], standards are also evolving to provide better support of the new architectures, pushing for more advanced compilation flows and runtime execution environments.

The current compiler technology is still based on a three-tier multiple-pass approach designed in the seventies to support different languages and targets at a reduced cost [44]. The front-end deals with the specificites of the input languages and generates an intermediate representation (IR) that is analyzed and optimized by a generic middle-end. Then the optimized intermediate representation is lowered to the target-machine level taking into account its instruction set architecture (ISA). Following the success of Java and leveraging the growing processing and memory resources of conventional hardware, just-in-time compilation schemes (JIT) have been added to the traditional static compiler technology to improve portability and response time. For instance, most OpenCL compilers are JITs. However, optimizing compiler technology still does not support parallelism and does not provide any safety guarantees for conventional programming models. The intermediate representation used by commercial and open source compilers, static or JIT, are sequential in essence. OpenMP annotations are first converted into run-time calls. Chapel programs are translated into C or C++ using run-time calls to express the parallel constructs. These calls, as well as MPI user calls, inhibit usual sequential optimizations because their side effects are unknown. And no parallel optimizations are possible.

On the research side, three attempts, at least, have been made to improve the compiler technology and support parallelism. Antoniu Pop at MINES ParisTech, in collaboration with ENS, tried to keep OpenMP annotations implicit during the first passes of GCC and to instantiate them later, but did not succeed within the time frame he had. Vivek Sarkar and his team [46,47,48,49] designed new parallel intermediate representations (PIR), at a high- or low-level (HPIR and LPIR), to compile Habanero-Java [15], a research programming language derived from X10 [6] and Java, with new analyses, such as “happens before”, and new transformations, such as barrier elimination or loop interchange, dedicated to parallel code. Also at MINES ParisTech, Dounia Khaldi, Francois Irigoin and Pierre Jouvelot have developped a technology, SPIRE, to transform intermediate representations (IR) into parallel intermediate representations (PIR) without breaking the sequential passes [29]. They have provided a formal semantics of the resulting PIRs and applied SPIRE to the PIPS compilation framework [22] to show its feasability, but they have not addressed the correctness of the existing passes nor the introduction of optimizations designed for parallel constructs and the mix of sequential and parallel constructs. Also, static single assignment (SSA) internal representations (IR) have been used to lower the complexity of several compiler passes [19] and several kinds of SSA have been introduced, with a great potential to support parallel targets.

Safety is currently not addressed by compilers but by static analyzers. They share the front-end parts of compilers, but their internal representations are designed to support analyses rather than transformations by preserving as much semantic information as possible. So high-level intermediate representations are sought, but usually not found, in compiler IRs, which are almost always built on a control-flow graph (CFG) and simplified statements and variables. Multithreaded programs can now be analyzed [35], but it would be useful to address other parallel constructs found in user codes and new languages.

Objectives

The fundamental objective of ACOPAL is to define a parallel internal representation that is well founded theoretically, with a formal semantics, that is able to support a wide range of parallel and sequential languages, and that is useful to support both the traditional passes developed for sequential languages and new analyses and transformations required to optimize parallel code.

If we succeed in reaching that ambitious objective, the traditional language-independent middle-end could be enlarged to cover parallel languages. Moreover, the research on static analysis has been traditionnaly split between analyses for safety and analyses for optimization. Considering the respective advances in both fields, it seems relevant to study whether the solutions advocated by one side of the community (including choices of IR) are releveant to the other side.

The second objective is to see how two parallel internal representations could be used as primary input and output of the middle-end, with a high-level representation (HPIR) as input to provide as much semantic information as possible to the analysis passes of the middle-end, and a low-level representation (LPIR), for instance based on gated variants of the SSA form, to simplify as much as possible the code generation passes of the back-end. We also aim at using the same HPIR in static analyzers and at improving analyses using knowledge about the property of its parallel constructs. For instance, a parallel loop, such as for(i=0;i<n;i++) a[i]=b[i];, may define a quantified invariant over arrays, here *forall i in [0..n[a[i]=b[i]* .

Finally, we aim at implementing the new HPIR in the PIPS compilation framework, which is developped at MINES ParisTech, to demonstrate the usefulness and the practicality of this new internal representation with respect to pre-existing passes and to at least one new transformation pass dedicated to parallel constructs.

State of the Art

The InsPIRe representation, developed for the Insieme compiler [24], has been designed for multiple languages, C, C++, OpenMP, MPI and OpenCL, but the parallel constructs are encoded by built-ins, which is to be avoided to develop analyses and transformations for parallel code.

The pioneering work by J. Zhao, entitled *Habanero-Java (HJ) language and Parallelism. Intermediate Representation design* (http://design.cs.iastate.edu/vmil/2011/slides/p07-zhao-slides.pdf), shows that analyses and transformations [10,11] of a parallel internal representation are possible, but his representations are directly linked to only one language, Habanero Java.

The PLASMA internal representation [41] abstracts data parallelism and vector instructions but is limited to heterogeneous SIMD architectures.

The SPIRE methodology has been designed for multiple parallel languages and applied to three different internal representations, LLVM's [33], PIPS's [21,17] and WHIRL's [2]. It is a starting point for us, but we believe that it still uses too many run-time calls too soon, which disturbs analysis passes, and that its formal semantics could be extended with a revisited memory model and implicit memory transfers.

There are profound similarities between dataflow languages and the static single-assignment form (SSA) used in mainstream imperative compilation. Drawing from this observation, we aim to improve SSA to better express concurrency, while retaining the ability to generate efficient code. Previous attempts have failed to exploit this similarity. As we work towards our goal, memory, state, and other aspects intrinsic to imperative programming languages will certainly present great challenges.

The static analysis of the safety of programs has been mainly concerned with sequential programs [12] and only recently started focussing on parallel programs (for instance at ENS) as now even critical systems become multi-core [35]. Current parallel program analyses are however limited to the multithreaded model of parallel execution. Similarly to the case of compilation, encoding parallel constructs as opaque built-ins [34], while possible, has an adverse effect on the effectiveness (as well as the generality) of the analysis. Program analysis would greatly benefit from an IR exposing parallel constructs at the same level as other language operations in a semantically significant and an economical way.

Research Directions

Five research goals should be reached for the project to be fully successful: 1) definition of a new high-level parallel representation (NHPIR), 2) definition of a new low-level parallel representation (NLPIR), 3) development of static analyses on the NHPIR, 4) development of optimizing passes on the NHPIR, and 5) compatibility between the NHPIR and the NLPIR.

1. New high-level parallel internal representation in PIPS

The current PIR in PIPS has been developed successfully using the SPIRE methodology. But, as explained above, too many built-in calls are introduced too early, preventing the effective interleaving of many passes. In particular, we need a mechanism to express communications in an implicit manner rather than thru send/receive pairs which are hard to analyze. This work has been started by Nelson Lossing as part of his PhD work (Oct. 2013-Sept. 2016). We also want to check that unstructured parallel constructs, expressing task parallelism, are usable to avoid synchronizations by events and built-in calls. Finally, we need to check that PGAS [45] and streaming languages are properly covered. Work on streaming asynchronous languages [16] has been started by Pierre Guillou for his PhD (Oct. 2013-Sept. 2016). CRI has also experience with synchronous models of stream-based programming languages, in particular DSLs for audio processing programs. These applications are very computing-intensive, and are good candidates for parallel implementations (see, for instance, Berkeley's roadmap paper for parallel computing (http://dl.acm.org/citation.cfm?id=1562783). Looking at synergies in representing efficiently these various models should be a fruitful endeavor.

Postdoc researchers with a few years of experience in parallel languages and/or compilation of a parallel language and/or static analysis will have the knowledge necessary to coordinate these efforts. Also a new PhD student (Oct. 2015-Sept. 2018) will fill in the gaps, especially as PGAS languages are concerned.

This work will be regularly presented to ENS Antique team in order to integrate all useful constructs already developed for the static analysis of parallel code and to make sure that all constructs introduced in NHPIR are compatible with static analysis techniques.

This work is expected to be joint work by the three partners, ENS Antique, ENS Parkas and MINES ParisTech. The later will lead. A joint ACOPAL seminar will be key to keep all participants aware and critical of the decisions made.

2. New SSA-based low-level parallel internal representation

The static single-assignment form, or SSA, has evolved to become the dominant intermediate representation in modern compilers for imperative languages. LLVM was built from scratch around a well-defined intermediate language in SSA form. GCC has had TreeSSA merged into the main branch in 2004. To our knowledge, most current optimizing compilers now use SSA in some place or another. SSA essentially makes explicit the flow of data in a sequential program. It does so by forcing variables to be renamed into multiple versions. One assignment, one variable (version). But SSA is still very limited as far as memory effects are concerned, and it also brings together control flow and data flow information rather than entirely relying on data flow constraints. Both limitations restrict the usage of SSA to the analysis of scalar programs. Extensions to arrays and « gated » variants of the SSA have been proposed, but are far from full replacement [40,38,14]. In particular, they do not act as concrete intermediate representations amenable to program transformations and enabling the generation of efficient imperative code, with the exception of the loop-closed SSA variant designed at MINES ParisTech by Sebastian Pop and Pierre Jouvelot for which a formal semantics was proposed (the first for an SSA variant).
We propose to collaborate on evolutions of the latter. And we also wish to consider a longer term approach, leveraging the deep semantic relations between SSA and purely functional langages, and between the principles of gated SSA with data-flow synchronous langages. The latter form a subclass of so-called Kahn process networks, which can be seen alternatively as functional langages on unbounded « streams », or as graphs of processes communicating through blocking FIFO buffers. The synchronous subclass of Kahn networks is popular as a semantic basis for synchronous reactive languages, which have achieved tremendous success in the design and modeling of safety-criticial, concurrent applications [26,21,12,9]. The Parkas group at ENS specializes on the study and on the compilation of such langages. There is a solid interest in using synchronous reactive langages and their associated code generation algorithms as a starting point for a parallel intermediate representation. In particular, relaxed extensions of the synchronous principle, introduced in the Parkas group, allow to generate efficient nested loop code, which is not the case of conventional synchronous languages.

This work is joint work by the two partners, ENS Parkas and MINES ParisTech. The former will lead.

3. Development of static analyses on the NHPIR

Many static analyses have already been developed in PIPS for its sequential internal representation. The first step is to either make sure that they are still well-founded for the parallel constructs, or to extend them as necessary.

The second step is to introduce at least one new analysis specific for parallel languages. The analysis or analyses will be chosen when Nelson Lossing and/or Pierre Guillou will have made more progress in their own PhD works.

It would also be interesting to check that the parallel analysis/es developed by ENS Antique can be expressed easily on the NHPIR. The postdoc researcher(s) and the new PhD student could contribute.

4. Development of optimizing passes on the NHPIR

The goals are identical for transformations as for analyses but it is useful to distinguish between them in term of deliverables and impact. Transformations are more difficult because we want to keep the resulting internal representation compatible with all other analyses and transformations.

Many transformation passes have already been developed in PIPS for its sequential internal representation. The first step is to either make sure that they are still well-founded for the parallel constructs, or to extend them as necessary. The second step is to introduce at least one new transformation specific for parallel languages. The transformation or transformations will be chosen when Nelson Lossing and/or Pierre Guillou will have made more progress in their own PhD works. The postdoc researcher(s) and the new MINES ParisTech PhD student should contribute.

5. Compatibility between the NHPIR and the NLPIR

The new low-level parallel intermediate representation will be developped mostly by ENS Parkas in cooperation with MINES ParisTech CRI. The compatibility of the two representations will be ensured by this cooperation. Ideally, we would like the NLPIR to be a subset of the NHPIR so as to merge both in a unique multilevel PIR, with compatibility for most passes, analyses or transformations, but not always the same efficiency with lower-level representations.

If the integration turns out to be impossible, or useless, or too costly in man*months, we will propose a unique transformation from NHPIR to NLPIR. Formal techniques can be used to ensure the semantical correctness of such a transformation. CRI has some expertise in this regard, in particular regarding automated or assisted theorem proving with tools such as CoQ. This line of work is done in cooperation wih the Deducteam project at INRIA Paris.

This work is joint work by the two partners, ENS Parkas and MINES ParisTech. The latter will lead.

People involved in the project :

Prénom NOM	Statut	Laboratoire ou structure	Nom de l'équipe
Corinne Ancourt	Senior Researcher	MINES ParisTech	CRI
Ulysse Beaugnon	PhD Student	ENS Ulm	Parkas
Fabien Coelho	Senior Researcher	MINES ParisTech	CRI
Albert Cohen	Senior Research Scientist	INRIA/ENS Ulm	Parkas
Adrien Guatto	PhD Student	ENS Ulm	Parkas
Pierre Guillou	PhD Student	MINES ParisTech	CRI
Francois Irigoin	Senior Researcher	MINES ParisTech	CRI
Pierre Jouvelot	Senior Researcher	MINES ParisTech	CRI
Nelson Lossing	PhD Student	MINES ParisTech	CRI
Nhat Minh Lê	PhD Student	ENS Ulm	Parkas
Antoine Mine	Senior Researcher	CNRS/ENS Ulm	Antique
Marc Pouzet	Professor	ENS Ulm	Parkas
To be recruited in 2015	PhD Student	MINES ParisTech	CRI

References

1. Cilk 5.4.6 Reference Manual. Supercomputing Technologies Group, Massachusetts Institute of Technology Laboratory for Computer Science, Nov. 2001. URL http://supertech.lcs.mit.edu/cilk/manual-5.4.6.pdf.
2. Open64 Compiler: WHIRL Intermediate Representation. www.mcs.anl.gov/OpenAD/open64A.pdf, 2007.
3. MPI: A Message-Passing Interface Standard.http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf, Sep 2012.
4. OpenCL, The open standard for parallel programming of heterogeneous systems, https://www.khronos.org/opencl/
5. OpenSHMEM Application Programming Interface (version 1.0).http://upc.gwu.edu/documentation.html, 2012.
6. X10 Language Specification, Feb. 2013.URL http://x10.sourceforge.net/documentation/languagespec/x10-latest.pdf.
7. OpenMP Application Program Interface. http://www.openmp.org/mp-documents/OpenMP_4.0_RC2.pdf, Mar. 2013.
8. Chapel Language Specification 0.796, Oct 21, 2010. URL http://chapel.cray.com/spec/spec-0.796.pdf.
9. M. Alras, P. Caspi, A. Girault, and P. Raymond. Model-based design of embedded control systems by means of a synchronous intermediate model. In Design and Test in Europe (DATE), May 2009.
10. R. Barik, J. Zhao, D. Grove, I. Peshansky, Z. Budimlic, V. Sarkar. Communication Optimizations for Distributed-Memory X10 Programs. 25th IEEE International Parallel and Distributed Processing Symposium (IPDPS), May 2011.
11. R. Barik, J. Zhao, V. Sarkar. Interprocedural Strength Reduction of Critical Sections in Explicitly-Parallel Programs. The 22nd International Conference on Parallel Architectures and Compilation Techniques (PACT), September 2013.

12. D. Biernacki, J.-L. Colaco, G. Hamon, and M. Pouzet. Clock-directed Modular Code Generation of Synchronous Data-flow Languages. In ACM International Conference on Languages, Compilers, and Tools for Embedded Systems, June 2008.
13. B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, X. Rival. A static analyzer for large safety-critical software. In Proc. of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'03), 196–207, San Diego, California, USA, Jun. 2003. ACM.
14. B. Boissinot, A. Darte, F. Rastello, B. D. de Dinechin, and C. Guillon. Revisiting out-of-sea translation for correctness, code quality and efficiency. In Proceedings of the 7th annual IEEE/ACM International Symposium on Code Generation and Optimization, pages 114–125, 2009.
15. V. Cavé , J. Zhao, and V. Sarkar. Habanero-Java: the New Adventures of Old X10. In 9th International Conference on the Principles and Practice of Programming in Java (PPPJ), Aug 201
16. Y. Choi, Y. Lin, N. Chong, S. Mahlke, and T. Mudge. 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, 2009.
17. F. Coelho, P. Jouvelot, C. Ancourt, and F. Irigoin. Data and Process Abstraction in PIPS Internal Representation. In F. Bouchez, S. Hack, and E. Visser, editors, Proceedings of the Workshop on Intermediate Representations, pages 77–84, 2011.
18. T. U. Consortium. http://upc.gwu.edu/documentation.html, Jun 2005.
19. R. Cytron, J. Ferrante, B. K. Rosen, M. N. Wegman, and F. K. Zadeck. Efficiently Computing Static Single Assignment Form and the Control Dependence Graph. ACM Trans. Program. Lang. Syst., 13(4):451–490, Oct. 1991.
20. P. Guillou, F. Coelho, F. Irigoin, Automatic Streamization of Image Processing Applications, LCPC 2014, Portland (OR), USA
21. N. Halbwachs, P. Raymond, and C. Ratel. Generating efficient code from data-flow programs. In Programming Language Implementation and Logic Programming, volume 528 of Lecture Notes in Computer Science, pages 207–218, 1991.
22. F. Irigoin, P. Jouvelot, and R. Triolet. Semantical Interprocedural Parallelization: An Overview of the PIPS Project. In ICS, pages 244–251, 1991.
23. S. Jagannathan, V. Laporte, G. Petri, D. Pichardie and J. Vitek, Atomicity Refinement for Verified Compilation, PLDI '14 Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 27-27 . Also TOPLAS, July 2014, V. 36, n. 2, pp. 6.1--6.30
24. H. Jordan, S. Pellegrini, P. Thoman, K. Kofler, and T. Fahringer. INSPIRE: The Insieme Parallel Intermediate Representation. In Proceedings of the 22Nd International Conference on Parallel Architectures and Compilation Techniques, PACT'13, pages 7–18, Piscataway, NJ, USA, 2013. IEEE Press.
25. P. Jouvelot and R. Triolet. Newgen: A Language Independent Program Generator. Technical report, CRI/A-191, MINES ParisTech, Jul 1989.
26. G. Kahn. The semantics of a simple language for parallel programming. In Information processing, pages 471–475, Aug. 1974.
27. H. Kasahara, H. Honda, A. Mogi, A. Ogura, K. Fujiwara, and S. Narita. A Multi-Grain Parallelizing Compilation Scheme for OSCAR (Optimally Scheduled Advanced Multiprocessor). In Proceedings of the Fourth International Workshop on Languages and Compilers for Parallel Computing, pages 283–297, London, UK, 1992.
28. D. Khaldi. Automatic Resource-Constrained Static Task Parallelization – A Generic Approach–. PhD thesis, MINES ParisTech, Nov. 2013.
29. D. Khaldi, P. Jouvelot, C. Ancourt, and F. Irigoin. Task Parallelism and Data Distribution: An Overview of Explicit Parallel Programming Languages. In H.

Kasahara and K. Kimura, editors, LCPC, volume 7760 of Lecture Notes in Computer Science, pages 174–189, 2012.
30. Dounia Khaldi, Pierre Jouvelot, François Irigoin et Corinne Ancourt, SPIRE : A Methodology for Sequential to Parallel Intermediate Representation Extension, HiPEAC Computing Systems Week, Paris : France (2013)
31. The OpenCL Specification. Khronos OpenCL Working Group, Nov. 2012. URL http://www.khronos.org/registry/cl/specs/opencl-1.2.pdf.
32. C. Liao, O. Hernandez, B. Chapman, W. Chen, and W. Zheng. OpenUH: An Optimizing, Portable OpenMP Compiler. Concurr. Comput. : Pract. Exper., 19(18):2317–2332, Dec. 2007.
33. The LLVM Reference Manual (Version 2.6). The LLVM Development Team, http://llvm.org/docs/LangRef.html, Mar. 2010.
34. J. Merrill. GENERIC and GIMPLE: a New Tree Representation for Entire Functions. In GCC developers summit 2003, pages 171–180, 2003.
35. A. Miné. Static analysis of run-time errors in embedded real-time parallel C programs. In Logical Methods in Computer Science (LMCS), 8(1:26), 63 pages, Mar. 2012.
36. V. K. Nandivada, J. Shirako, J. Zhao, V. Sarkar , A Transformation Framework for Optimizing Task-Parallel Programs. ACM Transactions on Programming Languages and Systems (TOPLAS), Volume 35, May 2013.
37. D. Novillo. OpenMP and Automatic Parallelization in GCC. In the Proceedings of the GCC Developers Summit, Jun 2006.
38. D. Novillo. Memory SSA - a unified approach for sparsely representing memory operations. In Proceedings of the 2007 GCC Developers' Summit, Ottawa, Canada, July 2007.
39. R. W. Numrich and J. Reid. Co-array Fortran for Parallel Programming. SIGPLAN Fortran Forum, 17:1–31, Aug 1998. ISSN 1061-7264
40. K. J. Ottenstein, R. A. Ballance, and A. B. MacCabe. The program dependence web: a representation supporting control-, data-, and demand-driven interpretation of imperative languages. In Proceedings of the ACM SIGPLAN 1990 conference on Programming language design and implementation, pages 257–271, 1990.
41. S. Pai, R. Govindarajan, and M. J. Thazhuthaveetil. PLASMA: Portable Programming for SIMD Heterogeneous Accelerators. In Workshop on Language, Compiler, and Architecture Support for GPGPU, held in conjunction with HPCA/PPoPP 2010, Bangalore, India, Jan 9, 2010.
42. R. Raman, J. Zhao, V. Sarkar, M. Vechev, E. Yahav. Scalable and Precise Dynamic Data Race Detection for Structured Parallelism. 33rd ACM SIGPLAN conference on Programming Language Design and Implementation (PLDI), June 201
43. V. Sarkar and B. Simons. Parallel Program Graphs and their Classification. In U. Banerjee, D. Gelernter, A. Nicolau, and D. A. Padua, editors, LCPC, volume 768 of Lecture Notes in Computer Science, pages 633–655. Springer, 1993. ISBN 3-540-57659-2.
44. J. Stanier and D. Watson. Intermediate Representations in Imperative Compilers: A Survey. ACM Comput. Surv., 45(3):26:1–26:27, July 2013.
45. K. Yelick, D. Bonachea, W.-Y. Chen, P. Colella, K. Datta, J. Duell, S. L. Graham, P. Hargrove, P. Hilfinger, P. Husbands, C. Iancu, A. Kamil, R. Nishtala, J. Su, W. Michael, and T. Wen. Productivity and Performance Using Partitioned Global Address Space Languages. In Proceedings of the 2007 International Workshop on Parallel Symbolic Computation, PASCO '07, pages 24–32, New York, NY, USA, 2007. ACM.
46. J. Zhao and V. Sarkar. Intermediate Language Extensions for Parallelism. In Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11,

AGERE!'11, AOOPES'11, NEAT'11, VMIL'11, SPLASH '11 Workshops, pages 329–340, New York, NY, USA, 2011. ACM.

47. J. Zhao, J. Shirako, K. V. Nandivada, V. Sarkar. Reducing Task Creation and Termination Overhead in Explicitly Parallel Programs. The Nineteeth International Conference on Parallel Architectures and Compilation Techniques (PACT), September 2010.
48. J. Zhao, Habanero-Java (HJ) language and Parallelism. Intermediate Representation design.http://design.cs.iastate.edu/vmil/2011/slides/p07-zhao-slides.pdf
49. J. Zhao, V. Sarkar. Intermediate Language Extensions for Parallelism . 5th Workshop on Virtual Machine and Intermediate Languages (VMIL'11), October 2011.