Ragnarok: RAndom Graphs Never ARe OK

James A. Jablin
Brown University
Providence, RI 02912
jjablin@cs.brown.edu

Maurice Herlihy
Brown University
Providence, RI 02912
mph@cs.brown.edu

Abstract

Many researchers test graph scheduling algorithms on "random" graphs. We show that many of the random graphs used in the literature are not representative of actual programs by comparing random graphs to graphs derived from the SPECINT2000 benchmarks. We introduce Ragnarok (RAndom Graphs Never ARe OK), a new graph benchmark suite embodying characteristics of SPECINT2000 programs. We hope this benchmark suite will be used instead of random graphs.

1 Introduction

The process of mapping tasks to processors and the determination of start times for each task is called task scheduling. Subtask decomposition, the process of decomposing a program into subtasks connected by precedence constraints, facilitates task scheduling for programs. Precedence constraints define data and control dependences within the program. Subtask decomposition yields a task graph where nodes are tasks and edges are precedence constraints. A scheduler maps tasks to processors and determines start times for each task. Edges may be weighted to represent cost of communication between tasks. The assignment of tasks to processors and allocation of start times is called *scheduling*. Optimal scheduling is NP hard. Therefore heuristics are used to achieve efficient results.

Usually, scheduling is performed on directed acyclic graphs (DAGs). Representing program execution with a DAG misses opportunities for loop optimization. The execution of loops generally consumes the majority of the total execution time. Therefore, it is beneficial to represent loops in the graph and then optimize for them. Thus, prior work has demonstrated results on cyclic and acyclic task graphs.

In the past, no standard existed for either type of graph. Validating techniques required transforming codes into graphs by hand or randomly generating graphs with likely characteristics of real graphs. A wide

array of previous work generates results on random graphs $[1-70]$. Previous researchers assumed random graphs were characteristic of real program graphs. This assumption has never been validated. Without such validation, performance claims based on random graphs may be misleading; they may not translate into improved efficiency in real applications. Additionally, fair comparison between different techniques remains difficult due to the multitude and variety of evaluation methodologies.

We conduct a detailed analysis of random graphs compared to real program graphs. We conclude random graphs are unsuitable inputs for validation. Random graphs are easily constructed but are poor substitutes for real program graphs. Ragnarok (RAndom Graphs Never ARe OK), a new benchmark suite composed of real program graphs, allows researchers to evaluate techniques utilizing identical inputs with the guarantee of real applicable results.

Composed of program graphs from the SPECINT2000 benchmark suite, there are no random graphs in Ragnarok. To compare scheduling algorithms requires head-to-head comparison. When previous studies incorporate random graphs, it is unclear how to perform this comparison fairly. In the past researchers simply generated more random graphs, acquiescing to the fact that they were neither using identical inputs in their comparison nor testing on real inputs. This phenomenon leads to many disparate random graph generation methodologies, an environment not conducive to impartial head-to-head comparison or an intuitive evaluation methodology. Realistically, head-to-head comparison should be performed using identical inputs from related work. Validation on random graphs creates ambiguity as to whether the technique has real merit. Ragnarok allows fair head-to-head comparison with identical inputs by incorporating characteristics of real programs.

The paper outline is as follows. Section 2 provides background. Section 3 provides a detailed evaluation. Section 4 discusses related work. Finally, Section 5 concludes and Section 6 describes future work.

2 Background

The timeline provided in Figure 1 helps motivate the advantages of Ragnarok. Each data point represents a scheduling algorithm evaluated with random graphs. Connected points indicate researchers specifically citing a previous work's random graph generation methodology for their random graphs. As a whole there are few edges in the timeline. There exists a vast number of random graph generation methodologies, but no clear ancestral history tying all methodologies together. A large body of prior work uses unique methodologies, not citing any previous methodology. Low citations per methodology and the large number of single use methodologies is significant. No standard inputs exist for the community to rally behind. A lack of standardization impairs scientific effort. Attempts to compare scheduling algorithms reduce to unraveling the intricacies of experimental methodology for generating random graphs. No authors validate the use

of random graphs for evaluation. Scheduling jobs are not normally random, so it is confusing to test on uncommon random inputs.

All of the works cited use random graphs, not real graphs to evaluate their scheduling algorithms. In this paper, we show that random graphs do not display the same characteristics as real program graphs and are therefore unrepresentative. Using random graphs to evaluate techniques causes confusion and is misleading. These challenges can be overcome by using real program graphs found in Ragnarok.

Section 4 provides a brief survey of random graph generation methodologies in chronological order. To provide a complete survey would be impractical. Many methodologies are unique, never cited subsequently, and provide inadequate detail to reimplement.

Some techniques also supply a means of generating task execution time. Execution time will not be discussed. We concern ourselves with the connectedness or flow of the task graph. As such, all nodes and edges have unit execution time.

3 Results

In this section we introduce Ragnarok, a new standard benchmark suite of cyclic and acyclic task graphs from SPECINT2000 programs. Details of Ragnarok's construction are given. Additionally, we compare Ragnarok to traditional random graphs. Details of the random graph generator used in the evaluation are given as well. Every attempt was made to use a random graph generation methodology with commonalities with a majority of previous work.

3.1 Creating Random Task Graphs

Considering prior work we opt for a simple random graph technique comparable with previous techniques. The random graph generator takes two parameters: the number of nodes in the graph and the density (the number of edges).

Definition 1. *Given the number of vertices* $|V|$ *and density* $\in (0, 1]$*, the number of edges* $|E|$ *is*

$$|E| = density \times (|V|)^2.$$

We use density as the metric for number of edges to maintain parallels with prior work.

For this study, cyclic graphs were generated with between 5–500 nodes in increments of 5 and with density 0.05–0.60 in increments of 0.05. The random graph generator is an accurate reimplementation of

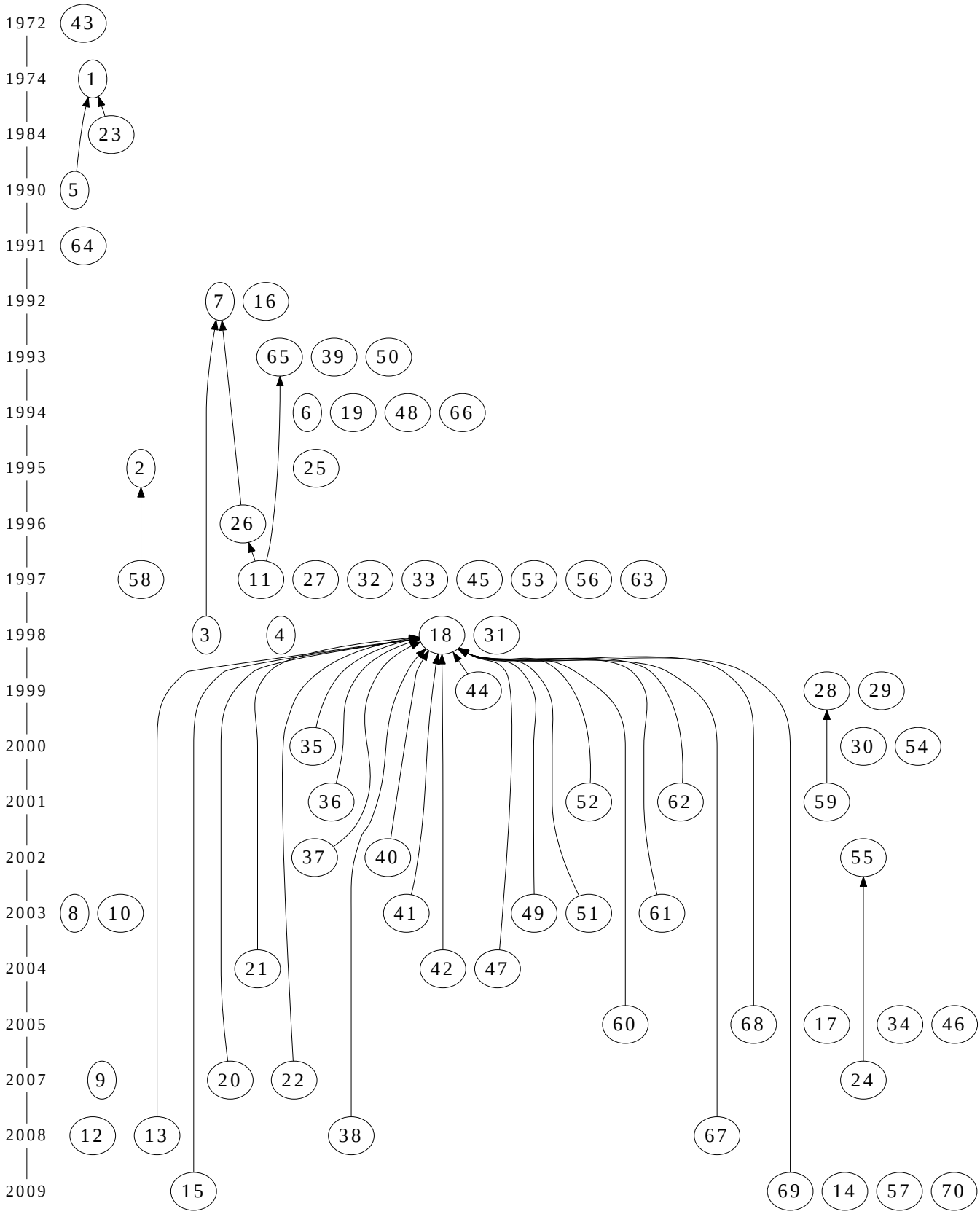


Figure 1: A timeline of random graph methodologies and citations.

```
1)  decyclify(Graph G)
2)      SCCs = strongly_connected_components(G)
3)      DAG_SCC = coalesce_SCCs(G, SCCs)
4)      return DAG_SCC
```

Figure 2: Algorithm to transform a cyclic graph into an acylic graph.

Sandnes and Sinnen's generator [46]. The parameters and the density range are the same. We generate graphs with up to 500 nodes while Sandnes and Sinnen stop at 100. With a density above 0.60, random graphs almost always form a single *strongly connected component* (SCC) composed of nearly all the nodes in the graph. Generating graphs with density beyond 0.60 does not increase the diversity of the data set. These parameters adequately capture the size and degree of random graphs described in prior work.

The generation of acyclic graphs proceeds in a similar fashion to cyclic graph generation with an extra step performed at the end. It is worth noting that DAGs, by definition, cannot have as many edges as a cyclic graph because they do not have back edges.

Definition 2. *Given the number of vertices $|V|$ and density $\in (0, 1]$, the maximum number of edges E in a DAG is*

$$|E| = \frac{|V| \times |V - 1|}{2}.$$

Consequently, choosing a lower upper bound for density has the added benefit of allowing one to use Definition 1 for both cyclic and acyclic graphs.

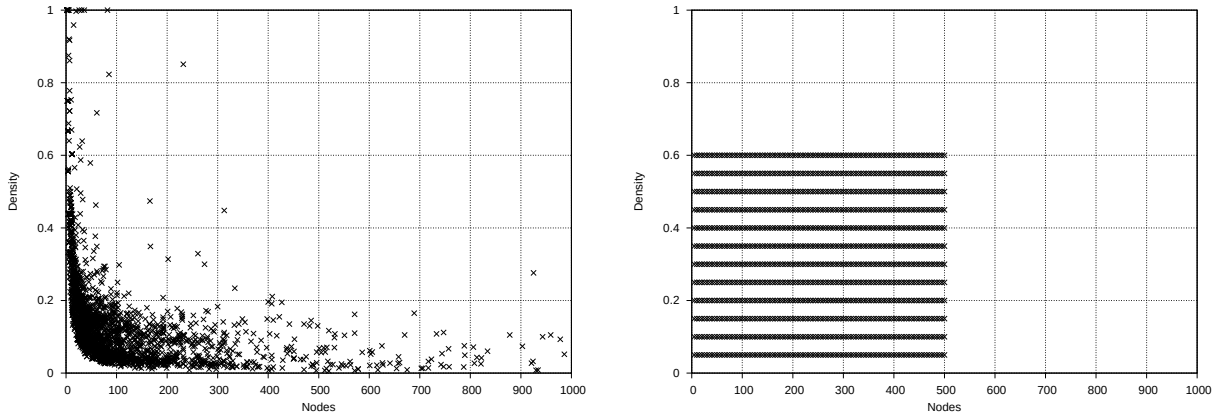
The extra step required for acyclic task graphs handles back edges. First, SCCs in the graph are identified. Then, for each SCC in the graph, we coalesce it into a single node. These steps are simple and solve the problem of cycles. We provide the pseudo-code for `decyclify` in Figure 2.

3.2 Ragnarok Benchmark

To create a new set of standard graphs that embody real characteristics of programs, we start with real programs rather than a random graph generator. We exploit a widely-used benchmark suite, the SPECINT2000 benchmark suite. Because it is tedious and error prone to transform programs into task graphs by hand, we use the LLVM compiler with the Clang front-end for translation. We translate with full alias analysis. Alias analysis more accurately characterizes program dependences.

These steps are all that is necessary for generating accurate cyclic task graphs from SPECINT2000 programs. To represent acyclic task graphs, we apply the `decyclify` algorithm. Other techniques to remove cycles by arbitrarily cutting edges is unsound.

3.3 Results for Acyclic Task Graphs



(a) Acyclic Ragnarok task graphs.

(b) Acyclic random task graphs.

Figure 3: Comparison of cyclic Ragnarok task graphs and random task graphs

Figure 3 contrasts acyclic random graphs and acyclic Ragnarok graphs. The random graphs were generated using the methodology similar to Yang and Gerasoulis [66]. Acyclic random graphs are characterized by a higher mode for the number of out edges. A higher modal score corresponds to a more strongly connected graph, reducing opportunities for parallelism, adversely affecting performance. Global connectedness throughout the program serializes execution. Acyclic random graphs are homogeneous in structure.

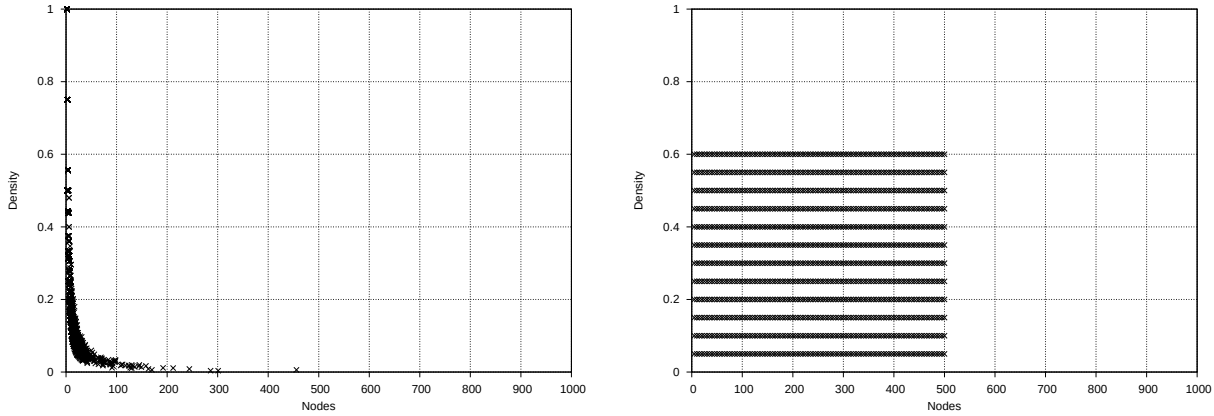
In contrast, real program graphs tend to have a lower mode, and therefore have a small number of nodes with very high and very low out degree. The variable density of different subgraphs within real graphs is not a normal distribution and defies random graph generation.

If acyclic random graphs are homogeneous, then real program graphs are heterogeneous. In practice, many real programs are known to contain parallelizable sections. Density in real program graphs is a local phenomenon, lowering the modal score. A lower modal score enables optimization opportunities. Real program graphs contain local diversity. Optimization leverages these natural opportunities for performance benefits.

3.4 Results for Cyclic Task Graphs

Results of comparing number of nodes and edges for cyclic graphs are shown in Figure 4. The generation methodology for cyclic random graphs parallels the technique in Sandnes and Sinnen [46]. The results are quite similar to the previous section. Considering the number of edges found in real programs, size of graph does not correlate to the number of edges. To the contrary, program graphs exhibit low density compared to random graphs. In real program graphs, larger graphs tend to have lower densities. Intuitively, programs

do not have large areas of task-parallelism, rather parallelism is locked inside SCCs. The SCCs represent loops and loop nests in the program.



(a) Cyclic Ragnarok task graphs.

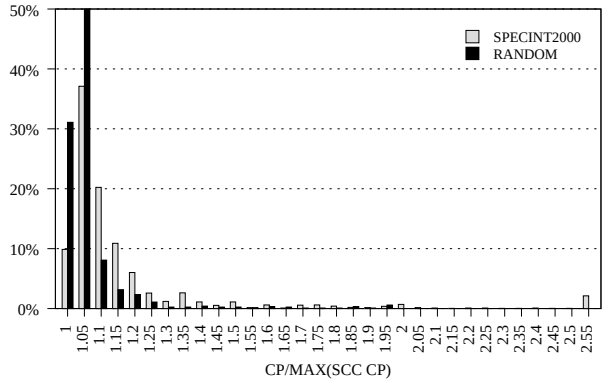
(b) Cyclic random task graphs.

Figure 4: Comparison of cyclic Ragnarok task graphs and random task graphs.

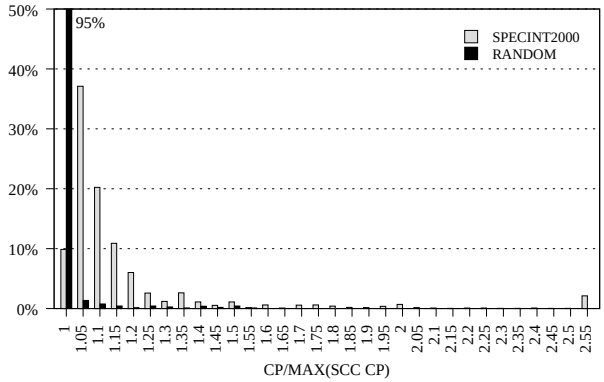
In order to diagnose whether cyclic random graphs are representative of real program graphs, we attempt to software pipeline both cyclic random graphs and cyclic Ragnarok graphs. Software pipelining is a graph-based compiler optimization that has demonstrated performance on program graphs. Software pipelining partitions a graph into subgraphs. Each subgraph becomes a stage in a pipeline, executing separately on its own processor and communicating its results to neighboring stages via a queue. Pipelining is most effective when stages are balanced. Pipelining achieves high performance because it maximizes throughput. Figure 5(a) shows the relationship between the size of the critical path for the largest SCC and the critical path for the whole program. A high ratio of $\frac{CP}{MAX(SCC\ CP)}$ strongly indicates increasing performance using software pipelining. Graphs with ratios close to 1.0 execute sequentially. The ratio therefore displays the raw ability of the graph to attain speedups using pipelining. Ragnarok is composed of many programs that can take advantage of software pipelining. On the other hand, random graphs show very little capability for software pipelining. Notice how the number of graphs with higher speedups abruptly stops after 1.05. Contrarily, Ragnarok graphs have a slow decrease in the number of graphs in each step to 2.0.

As indicated by Figure 4(a), random graphs are unrepresentative of real program structure. Thus, techniques utilizing real program structure fail on random graphs. An example of this fallacy is shown in Figure 5(a). Software pipelining shows potential performance benefits with Ragnarok graphs, but little to no speedup on random graphs. Consequently, techniques validated on random graphs may not show similar results on real program graphs.

Changing the parameters to the random graph generator does not help generate more "real" graphs. In Figure 5(b) the number of nodes and density for each cyclic Ragnarok graph was used to generate a

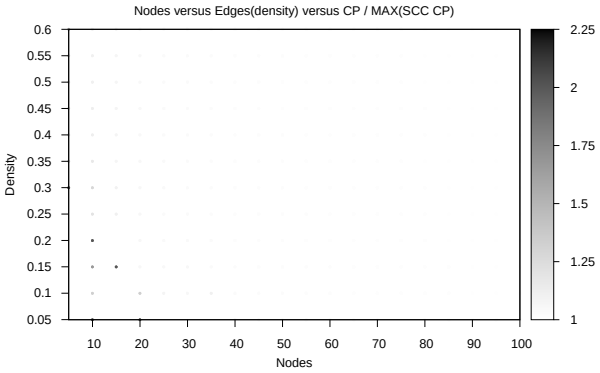


(a) Speedup with pipeline parallelism for cyclic task graphs.

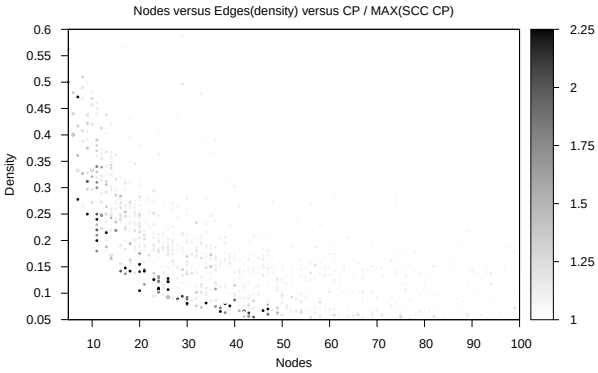


(b) Speedup with pipelin parallelism for cyclic task graphs: random graphs parameterized with Ragnarok values.

Figure 5: Parallelism exhibited by SCCs within cyclic task graphs and random task graphs.



(a) Cyclic random graphs.



(b) Cyclic Ragnarok graphs.

Figure 6: The effect of the number of nodes and density of cyclic graphs on pipeline parallelism.

corresponding cyclic random graph. The results in Figure 5(b) for random graphs are strictly worse than the results in Figure 5(a).

Complementing the results from Figure 5(a), Figure 6 helps characterize the types of graphs amenable to pipelining. As shown in Figure 6(a), only a sub-class of random graphs display the characteristics necessary for effective software pipelining. This sub-class includes random graphs with a very small number of nodes between 5 and 20 and densities of 0.05–0.30. As the number of edges grows quadratically with the number of nodes, large random graphs are characterized by a single SCC composed of a majority of the nodes. Real graphs, depicted in Figure 6(b), do not have these problems. Larger, more dense real graphs are amenable to pipelining. Random graphs are not representative of real graphs. Random graphs are unsuitable for validating scheduling algorithms.

Ragnarok enables easy validation of techniques for real inputs. Random graphs only guarantee demon-

strable performance benefits for random inputs. Ragnarok achieves the standardization and fair comparison required.

4 Related Work

Task graphs are most often modeled as DAGs. Previous work on scheduling assumes this acyclic representation of tasks and communication for scheduling purposes. Many languages provide some form of looping construct for iterative computation. Consequently, DAGs cannot accurately represent program structure. We first explore random acyclic task graph generation and subsequently move on to cyclic task graphs.

4.1 Random Acyclic Task Graphs

Adam et al. [1] generate random task graphs using four characteristics: (a) number of nodes, (b) height of the graph, (c) number of nodes at each level, and (d) average number of successors per node. Edges are randomly connected vertices from higher to lower levels. Edges must not violate this "high-to-lower" property lest the acyclic property be violated.

Almeida et al.'s technique favors a more probabilistic approach [7]. Random graphs have parameter π, representing the probability that two nodes are connected by an edge. Given an incidence matrix A with elements $a(i,j)$ representing a task graph with N nodes:

$$\begin{array}{lll} P[a(i,j)=0] = 1-\pi & \textbf{for} & 1 \leq i < j \leq N \\ P[a(i,j)=0] = 1 & \textbf{if} & i \geq j. \end{array} \quad (1)$$

Consequently, π serves a metric for the number of edges to insert into the graph. Task graphs exhibiting high parallelism will have π close to 1.0, while sequential task graphs will have π close to 0.0. Equation (1) prevents the formation of cycles.

Yang and Gerasoulis's approach to generating random task graphs compares favorably to Adam et al.'s [66]. Again Yang and Gerasoulis parameterize on number of nodes and number of levels. Instead of first choosing the number of nodes, the algorithm starts by randomly generating the number of levels. Then the number of nodes per level is chosen. Finally, nodes from disparate levels are connected by edges. It is assumed that some means of preventing cycles is imposed when inserting edges.

The methodology used in Ahmad and Kwok is very tersely described and thus the most mysterious. The authors control the number of nodes. No other details are given. With only this information, it would be hard to recreate this implementation. Wu and Shu [58] state they were provided the original implementation

of the random graph generator for their use.

TGFF provides a standard method for generating random task graphs [18]. It has fine-grained control of all conceivable attributes for composing nodes and edges. Publicly available, TGFF solves the problem of fair comparison between scheduling techniques. While exact recreation of a previous work's method for task graph generation may be impossible due to lack of details, TGFF supplies a shareable and reproducible set of tasks. Looking at the timeline in Figure 1, TGFF is the de facto standard tool for random graph generation. It is clearly the most cited. On the other hand, contemporaneous work exists that does not not cite it. Moreover, TGFF makes no effort to observe characteristics of real programs in its generated graphs. Not specifically created for validation purposes, TGFF only generates random graphs. The usefulness of these graphs is left open.

Strangely, we encounter a second random graph generator by Kwok and Ahmad [28]. This generator takes as input the number of nodes. The number of edges for each node is chosen from a uniform distribution with mean equal to $\frac{|V|}{10}$. Consequently, larger graphs have higher connectivity. Why this behavior is beneficial or valid is not discussed.

Similar to Adam et al.'s methodology, Topcuouglu et al. [55] generate task graphs using three parameters: (a) number of nodes N, (b) shape of the graph α, and (c) out degree of a node. The height of a graph is calculated from a uniform distribution with mean value equal to $\frac{\sqrt{N}}{\alpha}$. The width of each level is chosen from a uniform distribution with mean value equal to $\alpha(\sqrt{N})$. A short dense graph with high parallelism has α close to 1.0, whereas α close to 0 generates a long graph with low parallelism.

All the methodologies work similarly, but none use the same implementation. All the descriptions fail as adequate specifications for exact reimplementation; separate is inherently unequal. Fair comparison of scheduling algorithms cannot be made when no definitive standard exists.

Ragnarok alleviates the problem of standard inputs. The reimplementation of random graph methodologies becomes necessary. In addition, fair comparison can be performed easily. Ragnarok solves the problems in prior work.

4.2 Random Cyclic Task Graphs

Few previous works attempt to expose higher performance by utilizing a cyclic task graph representation. To statically schedule a cyclic graph, a "decyclifying" process must remove the cycles. Sandnes and Sinnen [46] implement a decyclification process to transform a cyclic task graph into a DAG. Their process is presented in two stages. In stage one, for each start node in a cyclic graph, construct a new acyclic graph by removing back edges. The *decyclify* algorithm behaves similarly to depth-first search. In stage two, search the constructed

DAGs for the graph with the shortest critical path.

Definition 3. *The length of a path p in $G = (\boldsymbol{V}, \boldsymbol{E}, w, c)$ is the sum of its nodes and edges:*

$$len(p) = \sum_{n \in p, \boldsymbol{V}} w(n) + \sum_{e \in p, \boldsymbol{E}} c(e).$$

Definition 4. *A critical path CP of a graph $G = (\boldsymbol{V}, \boldsymbol{E}, w, c)$ is a longest path in G.*

$$len(cp) = \max_{p \in G} \{len(p)\}.$$

Sandnes and Sinnen evaluate their *decyclify* algorithm on random cyclic task graphs. The two tunable parameters are the number of nodes and the graph density. Cyclic task graphs more successfully capture the flow of computation and communication inherent in the program.

Ragnarok's simple and straightforward methodology releases the researcher from the burden of fair comparison, few suitable inputs, and random graphs. Ragnarok contains program graphs from the SPECINT2000 benchmark. These programs provide a wide variety of characteristics found in other programs. Ragnarok enables fair comparison and a multitude of different inputs to facilitate standardization. Using random graphs for testing, it is both unclear and unlikely that results are applicable to real programs. Ragnarok fills the gap of uncertainty by providing real world inputs.

5 Conclusion

Random Task Graph generation is simple and creates a large set of inputs to test against. However, rather than preventing bias toward a particular scheduling algorithm, we conclude that the use of random graph generation creates an unrealistic experiment environment and a barrier for fair comparison between scheduling algorithms. Ragnarok incorporates characteristics of programs from SPECINT2000 benchmarks. Ragnarok achieves the necessary standardization to facilitate realistic and therefore useful comparison between disparate scheduling algorithms, while still incorporating a wide diversity of real program graph structure.

6 Future Work

In the near future, the Ragnarok Benchmark Suite of program graphs will be made publicly accessible via web site. We hope that Ragnarok will be used instead of random graphs for validating future work. A standardized benchmark would also facilitate comparison of related work. We also plan to reevaluate prior

work using Ragnarok. Certainly random graphs are a poor choice for validation, but most of the scheduling algorithms have merit, albeit less than initially advertised.

References

[1] Thomas L. Adam, K. M. Chandy, and J. R. Dickson. A comparison of list schedules for parallel processing systems. *Commun. ACM*, 17(12):685–690, 1974. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/361604.361619.

[2] Ishfaq Ahmad and Yu-Kwong Kwok. A parallel approach for multiprocessor scheduling. In *IPPS '95: Proceedings of the 9th International Symposium on Parallel Processing*, page 289, Washington, DC, USA, 1995. IEEE Computer Society. ISBN 0-8186-7074-6.

[3] Ishfaq Ahmad and Yu-Kwong Kwok. On exploiting task duplication in parallel program scheduling. *IEEE Trans. Parallel Distrib. Syst.*, 9(9):872–892, 1998. ISSN 1045-9219. doi: http://dx.doi.org/10.1109/71.722221.

[4] Ishfaq Ahmad and Yu-Kwong Kwok. Optimal and near-optimal allocation of precedence-constrained tasks to parallel processors: Defying the high complexity using effective search techniques. In *ICPP '98: Proceedings of the 1998 International Conference on Parallel Processing*, pages 424–431, Washington, DC, USA, 1998. IEEE Computer Society. ISBN 0-8186-8650-2.

[5] Mayez Al-Mouhamed and Adel Al-Maasarani. Performance evaluation of scheduling precedence-constrained computations on message-passing systems. *IEEE Trans. Parallel Distrib. Syst.*, 5(12): 1317–1321, 1994. ISSN 1045-9219. doi: http://dx.doi.org/10.1109/71.334905.

[6] Mayez A. Al-Mouhamed. Lower bound on the number of processors and time for scheduling precedence graphs with communication costs. *IEEE Trans. Softw. Eng.*, 16(12):1390–1401, 1990. ISSN 0098-5589. doi: http://dx.doi.org/10.1109/32.62447.

[7] V. A. F. Almeida, I. M. M. Vasconcelos, J. N. C. Árabe, and D. A. Menascé. Using random task graphs to investigate the potential benefits of heterogeneity in parallel systems. In *Supercomputing '92: Proceedings of the 1992 ACM/IEEE conference on Supercomputing*, pages 683–691, Los Alamitos, CA, USA, 1992. IEEE Computer Society Press. ISBN 0-8186-2630-5.

[8] Andy Auyeung, Iker Gondra, and H. K. Dai. Multi-heuristic list scheduling genetic algorithm for task scheduling. In *SAC '03: Proceedings of the 2003 ACM symposium on Applied computing*, pages 721–724, New York, NY, USA, 2003. ACM. ISBN 1-58113-624-2. doi: http://doi.acm.org/10.1145/952532.952673.

[9] Shahaan Ayyub and David Abramson. Gridrod: a dynamic runtime scheduler for grid workflows. In *ICS '07: Proceedings of the 21st annual international conference on Supercomputing*, pages 43–52, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-768-1. doi: http://doi.acm.org/10.1145/1274971.1274980.

[10] Savina Bansal, Padam Kumar, and Kuldip Singh. An improved duplication strategy for scheduling precedence constrained graphs in multiprocessor systems. *IEEE Trans. Parallel Distrib. Syst.*, 14(6): 533–544, 2003. ISSN 1045-9219. doi: http://dx.doi.org/10.1109/TPDS.2003.1206502.

[11] P. Bjorn-Jorgensen and J. Madsen. Critical path driven cosynthesis for heterogeneous target architectures. In *CODES '97: Proceedings of the 5th International Workshop on Hardware/Software Co-Design*, page 15, Washington, DC, USA, 1997. IEEE Computer Society. ISBN 0-8186-7895-X.

[12] Haijun Cao, Hai Jin, Xiaoxin Wu, Song Wu, and Xuanhua Shi. Dagmap: Efficient scheduling for dag grid workflow job. In *GRID '08: Proceedings of the 2008 9th IEEE/ACM International Conference on Grid Computing*, pages 17–24, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 978-1-4244-2578-5. doi: http://dx.doi.org/10.1109/GRID.2008.4662778.

[13] Po-Chun Chang, I-Wei Wu, Jyh-Jiun Shann, and Chung-Ping Chung. Etahm: an energy-aware task allocation algorithm for heterogeneous multiprocessor. In *DAC '08: Proceedings of the 45th annual Design Automation Conference*, pages 776–779, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-115-6. doi: http://doi.acm.org/10.1145/1391469.1391667.

[14] Yi-Jung Chen, Chia-Lin Yang, and Yen-Sheng Chang. An architectural co-synthesis algorithm for energy-aware network-on-chip design. *J. Syst. Archit.*, 55(5-6):299–309, 2009. ISSN 1383-7621. doi: http://dx.doi.org/10.1016/j.sysarc.2009.02.002.

[15] HaNeul Chon and Taewhan Kim. Timing variation-aware task scheduling and binding for mpsoc. In *ASP-DAC '09: Proceedings of the 2009 Asia and South Pacific Design Automation Conference*, pages 137–142, Piscataway, NJ, USA, 2009. IEEE Press. ISBN 978-1-4244-2748-2.

[16] Y.-C. Chung and S. Ranka. Applications and performance analysis of a compile-time optimization approach for list scheduling algorithms on distributed memory multiprocessors. In *Supercomputing '92: Proceedings of the 1992 ACM/IEEE conference on Supercomputing*, pages 512–521, Los Alamitos, CA, USA, 1992. IEEE Computer Society Press. ISBN 0-8186-2630-5.

[17] Mohammad Daoud and Nawwaf Kharma. Gats 1.0: a novel ga-based scheduling algorithm for task scheduling on heterogeneous processor nets. In *GECCO '05: Proceedings of the 2005 conference on*

Genetic and evolutionary computation, pages 2209–2210, New York, NY, USA, 2005. ACM. ISBN 1-59593-010-8. doi: http://doi.acm.org/10.1145/1068009.1068378.

[18] Robert P. Dick, David L. Rhodes, and Wayne Wolf. Tgff: task graphs for free. In *CODES/CASHE '98: Proceedings of the 6th international workshop on Hardware/software codesign*, pages 97–101, Washington, DC, USA, 1998. IEEE Computer Society. ISBN 0-8186-8442-9.

[19] E. S. H. Hou, N. Ansari, and H. Ren. A genetic algorithm for multiprocessor scheduling. *IEEE Trans. Parallel Distrib. Syst.*, 5(2):113–120, 1994. ISSN 1045-9219. doi: http://dx.doi.org/10.1109/71.265940.

[20] Pao-Ann Hsiung, Pin-Hsien Lu, and Chih-Wen Liu. Energy efficient co-scheduling in dynamically reconfigurable systems. In *CODES+ISSS '07: Proceedings of the 5th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, pages 87–92, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-824-4. doi: http://doi.acm.org/10.1145/1289816.1289840.

[21] Jingcao Hu and Radu Marculescu. Energy-aware communication and task scheduling for network-on-chip architectures under real-time constraints. In *DATE '04: Proceedings of the conference on Design, automation and test in Europe*, page 10234, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2085-5-1.

[22] Wei-Hsuan Hung, Yi-Jung Chen, Chia-Lin Yang, Yen-Sheng Chang, and Alan P. Su. An architectural co-synthesis algorithm for energy-aware network-on-chip design. In *SAC '07: Proceedings of the 2007 ACM symposium on Applied computing*, pages 680–684, New York, NY, USA, 2007. ACM. ISBN 1-59593-480-4. doi: http://doi.acm.org/10.1145/1244002.1244156.

[23] H. Kasahara and S. Narita. Practical multiprocessor scheduling algorithms for efficient parallel processing. *IEEE Trans. Comput.*, 33(11):1023–1029, 1984. ISSN 0018-9340. doi: http://dx.doi.org/10.1109/TC.1984.1676376.

[24] Sang Cheol Kim, Sunggu Lee, and Jaegyoon Hahm. Push-pull: Deterministic search-based dag scheduling for heterogeneous cluster systems. *IEEE Trans. Parallel Distrib. Syst.*, 18(11):1489–1502, 2007. ISSN 1045-9219. doi: http://dx.doi.org/10.1109/TPDS.2007.1106.

[25] Yu-Kwong Kwok and I. Ahmad. Bubble scheduling: A quasi dynamic algorithm for static allocation of tasks to parallel architectures. In *SPDP '95: Proceedings of the 7th IEEE Symposium on Parallel and Distributeed Processing*, page 36, Washington, DC, USA, 1995. IEEE Computer Society. ISBN 0-8186-7195-5.

[26] Yu-Kwong Kwok and Ishfaq Ahmad. Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors. *IEEE Trans. Parallel Distrib. Syst.*, 7(5):506–521, 1996. ISSN 1045-9219. doi: http://dx.doi.org/10.1109/71.503776.

[27] Yu-Kwong Kwok and Ishfaq Ahmad. Efficient scheduling of arbitrary task graphs to multiprocessors using a parallel genetic algorithm. *J. Parallel Distrib. Comput.*, 47(1):58–77, 1997. ISSN 0743-7315. doi: http://dx.doi.org/10.1006/jpdc.1997.1395.

[28] Yu-Kwong Kwok and Ishfaq Ahmad. Fastest: A practical low-complexity algorithm for compile-time assignment of parallel programs to multiprocessors. *IEEE Trans. Parallel Distrib. Syst.*, 10(2):147–159, 1999. ISSN 1045-9219. doi: http://dx.doi.org/10.1109/71.752781.

[29] Yu-Kwong Kwok and Ishfaq Ahmad. Benchmarking and comparison of the task graph scheduling algorithms. *J. Parallel Distrib. Comput.*, 59(3):381–422, 1999. ISSN 0743-7315. doi: http://dx.doi.org/10.1006/jpdc.1999.1578.

[30] Yu-Kwong Kwok and Ishfaq Ahmad. Link contention-constrained scheduling and mapping of tasks. *Cluster Computing*, 3(2):113–124, 2000. ISSN 1386-7857. doi: http://dx.doi.org/10.1023/A:1019076003163.

[31] Dingchao Li, Akira Mizuno, Yuji Iwahori, and Naohiro Ishii. Booking heterogeneous processor resources to reduce communication overhead. In *SAC '97: Proceedings of the 1997 ACM symposium on Applied computing*, pages 354–360, New York, NY, USA, 1997. ACM. ISBN 0-89791-850-9. doi: http://doi.acm.org/10.1145/331697.332309.

[32] Dingchao Li, Yuji Iwahori, and Naohiro Ishii. A recursive time estimation algorithm for program traces under resource constraints. In *SAC '98: Proceedings of the 1998 ACM symposium on Applied Computing*, pages 635–640, New York, NY, USA, 1998. ACM. ISBN 0-89791-969-6. doi: http://doi.acm.org/10.1145/330560.331005.

[33] Jing-Chiou Liou and Michael A. Palis. A comparison of general approaches to multiprocessor scheduling. In *IPPS '97: Proceedings of the 11th International Symposium on Parallel Processing*, pages 152–156, Washington, DC, USA, 1997. IEEE Computer Society. ISBN 0-8186-7792-9.

[34] G. Q. Liu, K. L. Poh, and M. Xie. Iterative list scheduling for heterogeneous computing. *J. Parallel Distrib. Comput.*, 65(5):654–665, 2005. ISSN 0743-7315. doi: http://dx.doi.org/10.1016/j.jpdc.2005.01.002.

[35] Jiong Luo and Niraj K. Jha. Power-conscious joint scheduling of periodic task graphs and aperiodic tasks in distributed real-time embedded systems. In *ICCAD '00: Proceedings of the 2000 IEEE/ACM*

international conference on Computer-aided design, pages 357–364, Piscataway, NJ, USA, 2000. IEEE Press. ISBN 0-7803-6448-1.

[36] Jiong Luo and Niraj K. Jha. Battery-aware static scheduling for distributed real-time embedded systems. In *DAC '01: Proceedings of the 38th annual Design Automation Conference*, pages 444–449, New York, NY, USA, 2001. ACM. ISBN 1-58113-297-2. doi: http://doi.acm.org/10.1145/378239.378553.

[37] Jiong Luo and Niraj K. Jha. Static and dynamic variable voltage scheduling algorithms for real-time heterogeneous distributed embedded systems. In *ASP-DAC '02: Proceedings of the 2002 Asia and South Pacific Design Automation Conference*, page 719, Washington, DC, USA, 2002. IEEE Computer Society. ISBN 0-7695-1441-3.

[38] Parth Malani, Prakash Mukre, Qinru Qiu, and Qing Wu. Adaptive scheduling and voltage scaling for multiprocessor real-time applications with non-deterministic workload. In *DATE '08: Proceedings of the conference on Design, automation and test in Europe*, pages 652–657, New York, NY, USA, 2008. ACM. ISBN 978-3-9810801-3-1. doi: http://doi.acm.org/10.1145/1403375.1403532.

[39] M. C. Murphy and D. Rotem. Multiprocessor join scheduling. *IEEE Trans. on Knowl. and Data Eng.*, 5(2):322–338, 1993. ISSN 1041-4347. doi: http://dx.doi.org/10.1109/69.219739.

[40] Juanjo Noguera and Rosa M. Badia. Dynamic run-time hw/sw scheduling techniques for reconfigurable architectures. In *CODES '02: Proceedings of the tenth international symposium on Hardware/software codesign*, pages 205–210, New York, NY, USA, 2002. ACM. ISBN 1-58113-542-4. doi: http://doi.acm.org/10.1145/774789.774831.

[41] Juanjo Noguera and Rosa M. Badia. System-level power-performance trade-offs in task scheduling for dynamically reconfigurable architectures. In *CASES '03: Proceedings of the 2003 international conference on Compilers, architecture and synthesis for embedded systems*, pages 73–83, New York, NY, USA, 2003. ACM. ISBN 1-58113-676-5. doi: http://doi.acm.org/10.1145/951710.951722.

[42] Juanjo Noguera and Rosa M. Badia. Multitasking on reconfigurable architectures: microarchitecture support and dynamic scheduling. *ACM Trans. Embed. Comput. Syst.*, 3(2):385–406, 2004. ISSN 1539-9087. doi: http://doi.acm.org/10.1145/993396.993404.

[43] C. V. Ramamoorthy, K. M. Chandy, and Mario J. Gonzalez. Optimal scheduling strategies in a multiprocessor system. *IEEE Trans. Comput.*, 21(2):137–146, 1972. ISSN 0018-9340. doi: http://dx.doi.org/10.1109/TC.1972.5008918.

[44] David L. Rhodes and Wayne Wolf. Overhead effects in real-time preemptive schedules. In *CODES '99: Proceedings of the seventh international workshop on Hardware/software codesign*, pages 193–197, New York, NY, USA, 1999. ACM. ISBN 1-58113-132-1. doi: http://doi.acm.org/10.1145/301177.301529.

[45] F. E. Sandnes and G. M. Megson. Improved static multiprocessor scheduling using cyclic task graphs: A genetic approach. In *Parallel Computing: Fundamentals, Applications and New Directions, North-Holland*, pages 703–710. Elsevier, North-Holland, 1997.

[46] Frode E. Sandnes and Oliver Sinnen. A new strategy for multiprocessor scheduling of cyclic task graphs. *Int. J. High Perform. Comput. Netw.*, 3(1):62–71, 2005. ISSN 1740-0562. doi: http://dx.doi.org/10.1504/IJHPCN.2005.007868.

[47] Marcus T. Schmitz, Bashir M. Al-Hashimi, and Petru Eles. Iterative schedule optimization for voltage scalable distributed embedded systems. *ACM Trans. Embed. Comput. Syst.*, 3(1):182–217, 2004. ISSN 1539-9087. doi: http://doi.acm.org/10.1145/972627.972636.

[48] C. Selvakumar and C. Siva Ram Murthy. Scheduling precedence constrained task graphs with non-negligible intertask communication onto multiprocessors. *IEEE Trans. Parallel Distrib. Syst.*, 5(3): 328–336, 1994. ISSN 1045-9219. doi: http://dx.doi.org/10.1109/71.277783.

[49] Dongkun Shin and Jihong Kim. Power-aware scheduling of conditional task graphs in real-time multiprocessor systems. In *ISLPED '03: Proceedings of the 2003 international symposium on Low power electronics and design*, pages 408–413, New York, NY, USA, 2003. ACM. ISBN 1-58113-682-X. doi: http://doi.acm.org/10.1145/871506.871607.

[50] G. C. Sih and E. A. Lee. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE Trans. Parallel Distrib. Syst.*, 4(2):175–187, 1993. ISSN 1045-9219. doi: http://dx.doi.org/10.1109/71.207593.

[51] Radoslaw Szymanek and Krzysztof Krzysztof. Partial task assignment of task graphs under heterogeneous resource constraints. In *DAC '03: Proceedings of the 40th annual Design Automation Conference*, pages 244–249, New York, NY, USA, 2003. ACM. ISBN 1-58113-688-9. doi: http://doi.acm.org/10.1145/775832.775895.

[52] Radoslaw Szymanek and Krzysztof Kuchcinski. A constructive algorithm for memory-aware task assignment and scheduling. In *CODES '01: Proceedings of the ninth international symposium on Hardware/software codesign*, pages 147–152, New York, NY, USA, 2001. ACM. ISBN 1-58113-364-2. doi: http://doi.acm.org/10.1145/371636.371706.

[53] Xinan Tang, J. Wang, Kevin B. Theobald, and Guang R. Gao. Thread partitioning and scheduling based on cost model. In *SPAA '97: Proceedings of the ninth annual ACM symposium on Parallel algorithms and architectures*, pages 272–281, New York, NY, USA, 1997. ACM. ISBN 0-89791-890-8. doi: http://doi.acm.org/10.1145/258492.258519.

[54] Sissades Tongsima, Timothy W. O'neil, Chantana Chantrapornchai, and Edwin H.-M. Sha. Properties and algorithms for unfolding of probabilistic data-flow graphs. *J. VLSI Signal Process. Syst.*, 25(3): 215–233, 2000. ISSN 0922-5773. doi: http://dx.doi.org/10.1023/A:1008187622838.

[55] Haluk Topcuouglu, Salim Hariri, and Min-you Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. Parallel Distrib. Syst.*, 13(3):260–274, 2002. ISSN 1045-9219. doi: http://dx.doi.org/10.1109/71.993206.

[56] Lee Wang, Howard Jay Siegel, Vwani R. Roychowdhury, and Anthony A. Maciejewski. Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach. *J. Parallel Distrib. Comput.*, 47(1):8–22, 1997. ISSN 0743-7315. doi: http://dx.doi.org/10.1006/jpdc.1997.1392.

[57] Xiaofeng Wang, Rajkumar Buyya, and Jinshu Su. Reliability-oriented genetic algorithm for workflow applications using max-min strategy. In *CCGRID '09: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 108–115, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-0-7695-3622-4. doi: http://dx.doi.org/10.1109/CCGRID.2009.14.

[58] Min-You Wu and Wei Shu. On parallelization of static scheduling algorithms. *IEEE Trans. Softw. Eng.*, 23(8):517–528, 1997. ISSN 0098-5589. doi: http://dx.doi.org/10.1109/32.624307.

[59] Min-You Wu, Wei Shu, and Jun Gu. Efficient local search for dag scheduling. *IEEE Trans. Parallel Distrib. Syst.*, 12(6):617–627, 2001. ISSN 1045-9219. doi: http://dx.doi.org/10.1109/71.932715.

[60] Liang Yang, Tushar Gohad, Pavel Ghosh, Devesh Sinha, Arunabha Sen, and Andrea Richa. Resource mapping and scheduling for heterogeneous network processor systems. In *ANCS '05: Proceedings of the 2005 ACM symposium on Architecture for networking and communications systems*, pages 19–28, New York, NY, USA, 2005. ACM. ISBN 1-59593-082-5. doi: http://doi.acm.org/10.1145/1095890.1095895.

[61] Peng Yang and Francky Catthoor. Pareto-optimization-based run-time task scheduling for embedded systems. In *CODES+ISSS '03: Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 120–125, New York, NY, USA, 2003. ACM. ISBN 1-58113-742-7. doi: http://doi.acm.org/10.1145/944645.944680.

[62] Peng Yang, Chun Wong, Paul Marchal, Francky Catthoor, Dirk Desmet, Diederik Verkest, and Rudy Lauwereins. Energy-aware runtime scheduling for embedded-multiprocessor socs. *IEEE Des. Test*, 18 (5):46–58, 2001. ISSN 0740-7475. doi: http://dx.doi.org/10.1109/54.953271.

[63] T. Yang and A. Gerasoulis. Dsc: Scheduling parallel tasks on an unbounded number of processors. Technical report, Santa Barbara, CA, USA, 1994.

[64] Tao Yang and Cong Fu. Heuristic algorithms for scheduling iterative task computations on distributed memory machines. *IEEE Trans. Parallel Distrib. Syst.*, 8(6):608–622, 1997. ISSN 1045-9219. doi: http://dx.doi.org/10.1109/71.595579.

[65] Tao Yang and Apostolos Gerasoulis. A fast static scheduling algorithm for dags on an unbounded number of processors. In *Supercomputing '91: Proceedings of the 1991 ACM/IEEE conference on Supercomputing*, pages 633–642, New York, NY, USA, 1991. ACM. ISBN 0-89791-459-7. doi: http://doi.acm.org/10.1145/125826.126138.

[66] Tao Yang and Apostolos Gerasoulis. List scheduling with and without communication delays. *Parallel Comput.*, 19(12):1321–1344, 1993. ISSN 0167-8191. doi: http://dx.doi.org/10.1016/0167-8191(93)90079-Z.

[67] Yang Yu and Viktor K. Prasanna. Energy-balanced task allocation for collaborative processing in wireless sensor networks. *Mob. Netw. Appl.*, 10(1-2):115–131, 2005. ISSN 1383-469X. doi: http://doi.acm.org/10.1145/1046430.1046440.

[68] Mingxuan Yuan, Xiuqiang He, and Zonghua Gu. Hardware/software partitioning and static task scheduling on runtime reconfigurable fpgas using a smt solver. In *RTAS '08: Proceedings of the 2008 IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 295–304, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 978-0-7695-3146-5. doi: http://dx.doi.org/10.1109/RTAS.2008.39.

[69] Ping-Hung Yuh, Chia-Lin Yang, Chi-Feng Li, and Chung-Hsiang Lin. Leakage-aware task scheduling for partially dynamically reconfigurable fpgas. *ACM Trans. Des. Autom. Electron. Syst.*, 14(4):1–26, 2009. ISSN 1084-4309. doi: http://doi.acm.org/10.1145/1562514.1562520.

[70] Yang Zhang, Charles Koelbel, and Keith Cooper. Hybrid re-scheduling mechanisms for workflow applications on multi-cluster grid. In *CCGRID '09: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 116–123, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-0-7695-3622-4. doi: http://dx.doi.org/10.1109/CCGRID.2009.60.