Slack Exploitation for Aggressive Dynamic Power Reduction in SoC

Aleksandar Milutinović
University of Twente
Enschede, The Netherlands
a.milutinovic@utwente.nl

Kees Goossens
NXP Research, Eindhoven &
Delft University of Technology
Delft, The Netherlands
kees.goossens@nxp.com

Gerard J.M. Smit
University of Twente
Enschede, The Netherlands
g.j.m.smit@ewi.utwente.nl

Abstract—**The increasing power consumption of today's system-on-chip (SoC) outpaces the trend of increasing battery capacity. The applications offered to customers grow tremendously too, a trend that is accelerating in the future. This yields stronger requirements for lower power consumption. During design, a system is dimensioned to worst-case workload requirements. Most of the time, workload is far below this level, which results in slack in some parts of the system. Our idea is to exploit this available slack by using adequate variants of dynamic voltage and frequency scaling and power gating. For scalability reasons, we commence our research with local dynamic adaptive power and frequency scaling, based on the slack observed at run time. This paper presents the motivations and possible directions for our research.**

Keywords—**system-on-chip, tiled architecture, dynamic power reduction, slack exploitation.**

I. Introduction

Handheld mobile electronic devices have been people's companions in everyday lives for decades now. As the number and their variety is growing rapidly nowadays, the trend goes towards fewer but more complex devices. Instead of many, ultimately there will be a single device with integrated all features and services desired by user, such as various communication services, entertainment and business applications.

This paper has following organization. We describe the scope of our research in Section II. The current trends related to design of SoC are outlined in Section III. In Section IV describes the problems addressed. The related work is discussed in Section V. The description of the proposed solution in Section VI. Finally, Section VIII concludes the paper.

II. Scope

An architecture in our scope is a low-power MPSoC for handheld devices, a heterogeneous multi-resource tiled architecture with different types of resources (Figure 1), including
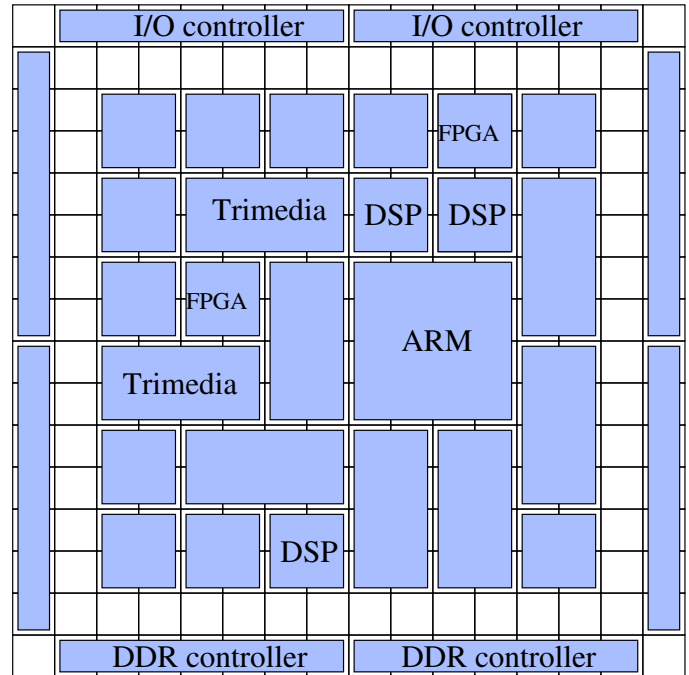1. processing – general purpose processors (e.g.

Fig. 1: Tiled architecture of heterogeneous MPSoC. figure

ARM, Trimedia), DSP, dedicated ASIC IPs, coarse grain reconfigurable processors (e.g. Montium [7]);
2. memory – on-chip memory (SRAM) and controllers for external memory;
3. I/O controllers, and
4. interconnect – tiles are connected through a network on chip [5] or using buses.

All tiles are power-manageable, to scale down and up their supply voltage and frequency, as well as switching the clock and power on and off. This should be done independently per resource or per power domain.

Applications in the scope are streaming applications, characterized by their periodic nature. Deadlines given by applications are the main guidelines for our power management solutions. Aperiodic applications are also in the scope, as they are common in the

systems today, but they will be treated as best-effort service. We assume that an application is composed of jobs, each of which is a set of multiple dependent tasks that are mapped on one or more tiles.

III. Trends

A. MPSoC

The general trend for SoC design is to build systems using reusable cores with more specialized IPs and connect them through a network on chip. Integration capabilities are constantly growing and it is certain that the number of tiles will be hundreds or even thousands. These manycores, as they are called, are presumably the solution for design issues raised by rapidly growing demands on the customer side.

B. Applications

The number of services and applications offered to the customer is growing for every product generation and new applications are released even more often. Applications like voice communication for example, lost their supremacy among end-users demands. A variety of multimedia and video communication applications will push system design towards higher requirements. Usually, within a single application there is more than one mode in which the application can run. Differences in processing requirements between the modes show considerable variations. This leads to slack time and not completely utilized resources. Additionally, the composition of tasks and dependencies within an application mode varies over time. The situation is worse when multiple applications and their modes should run together on the system. The number of these use-cases explodes and it is not feasible to analyze them in full detail at the design-time (as pointed out by Hansson [6]).

Another source of variation is the input data. It may influence and may lead to changes in application modes and also varies within the mode. A task consumes input data in data blocks, usually called data tokens that correspond to a sample or a video frame.

Processing requirements of successive data tokens generally varies while the amount of time given to it by worst-case execution time is constant. New video codecs and standards, for example, introduce more complex data transformations and differences in processing the same amount of data tokens are getting bigger. The same holds for wireless communication protocols.

Having in mind this variety of applications, the handheld devices are going in the direction of general purpose computers, but mobility and energy reasons keep them still in domain of embedded systems.

C. Transistors

Transistors are getting smaller and less reliable. Variations originating from the fabrication process dictate the performance of silicon during operation, and the difference between nominal and actual performance grows rapidly. The variations have a spatial component over a die. This results in a difference between actual and worst-case energy usage. In the current paradigm of design for the worst case, these variations result in an overdimensioned system, which is not fully utilized at run-time.

D. Power and Energy

As mobile devices are powered by batteries and the trend for battery capacity is outrun by the trends of energy consumption required by applications, research is paying more attention to possibilities for saving power. Dynamic power is increased due to the growing number of components and is relevant only when a core is active. On the other hand, energy of the battery is continuously drawn by static power while the component is powered on. Predictions for a few next coming process technologies claim that static power will increase to the amount which is comparable to dynamic power.

IV. Problems

In order to design a system capable of fulfilling all requirements, designers mostly take into account the worst-case requirements of applications, plus some extra performance as a margin for reliability and tolerance issues. The worst case typically happens very rarely and the actual case is usually using just a small part of available resources. This way the system is poorly utilized most of the time and unnecessarily burns energy more than needed when constantly run at maximal voltage and frequency. This is currently one of the limiting factors for accepting a product. Besides the energy waste, silicon area is not used very efficiently even if it is power gated. Thus, there are two directions in solving this problem, whether the system is optimized for energy or silicon area.

Power management (PM) aims to solve this problem. PM adjusts the operating voltage and frequency to the current work load and thus system operates more energy efficiently. But, this solves only the part of the problem. Slowing down and speeding up is not

instantaneously, so it is effective only if there is enough slack in the system and there is sufficient time to meet the deadline. In some cases this fine-grained slack can be accumulated and used later for the same or the next period. Generally, extra buffering is needed for this, which can have a considerable cost. Otherwise, not all the available slack is used by PM. Due to the input data part of variation in workload is not known at run time.

The other approach is to dimension the system to reduced worst-case requirements, and pay the price in the resulting quality degradation through deadline misses when worst case happens. This can be seen as trade-off between quality and available resources of the system and it is done at design time.

At the end we point out the complex trade-off between resources (silicon area), power and quality of service (QoS): optimize the execution for power, while providing a certain level of quality, which can be also dynamically changed during execution, in order to favorite either longer playtime or improve quality. The system has to be dimensioned with this feature in mind.

V. RELATED WORK

The speed of voltage and frequency scaling and power gating infrastructure, used by current power management implementations, limits the minimal amount of slack that can be used to save energy. Shorter periods of inactivity are not used and sometimes even not reported as slack. The general argument is that the amount of energy saved this way would be negligible. Nevertheless, when that happens regularly, e.g. in every iteration but with different duration, the cumulative amount of wasted energy is again considerable. The most common approach is to just accumulate slack and then use it as the whole. The on-chip digital power supply control described in [9] offers fast response time for changing supplied voltage. What is missing from this work is the policy of the PM that can fully utilize its speed capabilities.

The common method for observing system utilization is observing the utilization of the main general purpose processor. The operating system periodically checks the status whether the processor is busy or idling, and then counts the current period atomically as busy or idle. After averaging, it reports the system occupation. This happens in periods typically ranging from 1 ms to 10 ms, so it works very coarsely, and introduces certain inaccuracy in measurement and in observation of slack. We think that more accurate measurements, supported by hardware, would be beneficial to PM and bring opportunities to reduce power consumption. The notion of progress from the start to end of a task execution should provide early detection of slack produced by variable execution time.

One of the most common methodologies to improve the capabilities of PM is to speculate on the workload of upcoming tasks or their iterations. Most of the work done is based on the predictions of future workload based on run-time history and statistical profiling of the workload. The fact is that prediction cannot be always sufficiently accurate.

We point out two reasons for erring a prediction. The first one is the desired feature of PM that it has to be universal and thus completely decoupled from applications. Experiments show that optimal gain of PM is achieved if it reacts with the same periodicity as the application, and further when they are phase aligned. This synchronization is hard when applications do not give any information. The second reason is that a notion of progress would be beneficial if it exposes the slack not only just before deadline or after completion of a task. We intend to make use of slack which is fine grained, and which has been out of the scope of current PM methods.

Tools could provide considerable amount of information to PM. For example, Gheorghita [3] suggests a concept based on application scenarios and use-cases. The same author suggests automatic detection of the scenarios and their profiling and characterization [4]. The limitation is that only big differences in scenarios can be used for management because of inertia of the support for voltage and frequency scaling and switching infrastructure. On characterisation of the workload and determining the worst-case execution time of the task segments there are different approaches. Lee [8] uses hierarchical finite state machines and a synchronous data flow model to calculate the remaining workload in actual execution case at run-time. Azavedo [2] uses the compiler to place the checkpoints in program code at the boundaries of basic blocks. The checkpoints carry user-defined time constraints. At these points processor recalculates and changes the frequency and voltage. The overhead of calculation and taking proper action introduces high overhead in computation and in execution time of the task, if processor has to idle during the transition.

AbouGhazaleh [1] presents the collaboration between compiler and operating system for purposes of power management for real-time applications. The compiler inserts the instrumentation code into the

program code. This code evaluates the worst-case remaining cycles at run-time. Periodically, the interrupt service routine observes this number and changes the speed of the processor in order to consume the least amount of energy. This method requires intervals between interrupts to be sufficiently short to satisfy all the paths in control flow diagram, and because of that can affect the granularity and limit the power management effect.

As stated before, optimal savings are achievable when PM and applications are synchronized, so the workload and progress information used for future performance prediction within power management is correct and trustworthy. The phase mismatch can lead to incorrect prediction and decisions, and to increased number of missed deadlines. For that reason we think the application should provide support to power management in form of period and synchronization hints.

VI. Proposal

The main goal of the research is to maximize the playtime of a battery powered device. We propose dynamic reduction of power consumption by slowing down the system when it can afford longer execution at lower performance level or to shut down parts of the system when they are idle for a longer period. Our research will focus on saving dynamic energy as the primary goal, using the DVFS as well as clock and power gating, closely coupled to current processing workload progress and current performance.

Our intention is to provide hardware and software solutions that will efficiently support power management on the system level based on credits. The compiler will translate the deadlines and application hints into the credits which will be fed to the hardware support. The credits will be obtained by tools, compilers and profilers, with possibility for hints to be given directly by application programmers, also in form of credits.

Direct support for monitoring events of interest and closely observing the progress of a task will be another feature implemented by hardware support. That will improve early detection of slack, whether negative or positive and thus, prevent deadline misses or save power. Hardware that we propose will provide the support for quick and computationally nondemanding slack detection towards the just-on-time deadline completion.

The hardware part of proposed solution has two major components, as shown in Figure 2. The first
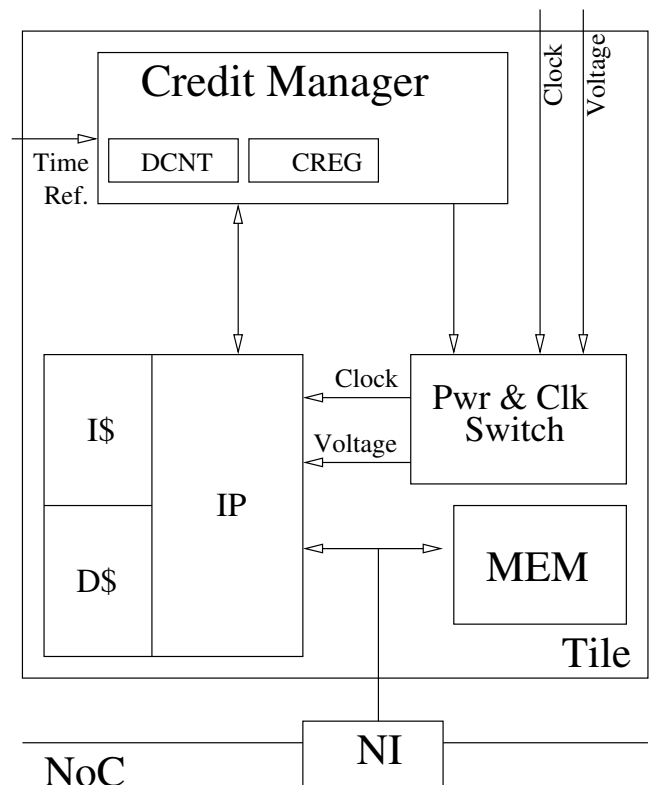

Fig. 2: Organization of a tile.

figure

component is the switching infrastructure for DVFS, power and clock gating. This is represented by Power and Clock Switch in Figure 2. This component is almost completely developed in many variants [9] and we will use it without major changes.

The second hardware component is the support for the credit based system, call the Credit Manager (CM). It receives credits from tiles, processes them and if there is a slack reacts in appropriate way by giving instructions to Power and Clock Switch. For these purposes the CM contains a deadline counter (DCNT), giving a notion of the time remaining until the deadline. The Cm also contains a credit register (CREG), which keeps track of the amount of workload to be done until the deadline. Every tile or domain will have a CM with the responsibility to: 1) receive credits and deadline indications from the IP, 2) estimate and adjust the voltage and frequency condition based on the workload and deadline, 3) report slack time, deadline misses or completion, and 4) optionally, report performance counters and slack remained after task completion.

The basic idea is that the compiler, during compilation, provides credits based on estimation and/or profiling of the program code. The credits will be re-

lated to cycle count needed by a certain segment of the program code. The sum of the credits given to the segments on the critical path in control flow graph of a task is equal to the amount of credits that represent the deadline. At the start of the code, the IP block reports to CM the total amount of credits for the task, and the task deadline. These are stored in the CREG and DCNT. The deadline counter is decremented after a certain period of time that represents the normalized value of one credit. After a segment is executed, the IP block will report amount of credits associated with the segment to CM. They will be subtracted from CREG. When the task finishes, the value of CREG is zero but the DCNT can contain zero or a positive value. When a segment uses its worst-case execution time (WCET) and just meets its deadline, the value of DCNT is zero. Otherwise, the task finishes before the deadline, and the value of deadline counter will be the slack generated during the task execution. The IP can get this information and use it according to its slack exploitation policy. The deadline counter value cannot be negative because the characterization of worst-case execution time is conservative, and actual-case execution time cannot exceed it. Ideally, if credit system is coupled with enabled power manager, as we show below, the slack at the deadline will be zero. The frequency calculation is the ratio of remained workload (the value of CREG) and the time left before the deadline (the value of DCNT).

The described credit system is able to use slack, for example from the following sources:
1. different task execution paths, e.g. different branches are executed;
2. workload variation, e.g. variable count of loop iterations;
3. variable I/O operation execution time;
4. variable instruction time due to the complexity of instructions and their operands, and
5. cache memory misses.

The following example shows one possible variant of the proposed credit system and the way the slack is exposed. Figure 3 shows an example C program; the corresponding control flow graph is shown in Figure 4. The edges of the graph represent the segments of the program code and the (fictitious) credits associated with them are given in parentheses. The nodes are the bounds of program code segments and they are associated with the amount of credits for subtraction when task reaches them. They have the maximal value of all incoming edges, e.g. for node 4 it is $\max\{C, G\} = 30$.

```
 1:  initialise();                  //A
 2:  i=10; sum=0;                    //A
 3:  while(a[i]>0 && i>0) {          //B F
 4:      sum+=a[i];                  //B F
 5:      i--;                        //B F
 6:  }
 7:  if (sum>max) {
 8:      x=sum+(sum-max)*rate;       //C
 9:      for(i=0;i<10;i++)           //C
10:         a[i]=a[i]*coef[i]-offset1; //C
11:  } else {
12       x=sum;                      //G
13:      for (i=0;i<10;i++)          //G
14:         a[i]-=offset2;           //G
15:  }
16:  output(x);                      //D
17:  cleanup();                      //E
```
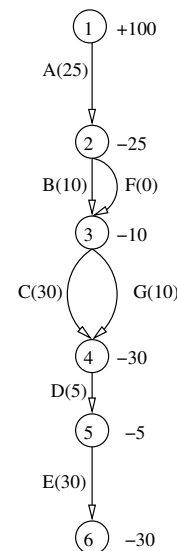
Fig. 3: Example C program.
figure

Fig. 4: Control flow graph of C program with credits.
figure

TABLE I: credit values for different executions.
table

node	critical path	non-critical path	workload variation
① (+100)	100	100	100
② (-25)	75	75	90 (+15)
③ (-10)	65	75 (+10)	85 (+5)
④ (-30)	35	65 (+20)	75 (+20)
⑤ (-5)	30	60	74 (+4)
⑥ (-30)	0	30	44

TABLE II: Using slack for DVFS.

table

node	WCET case CREG/DCNT	time	f	case 1 – no DVFS CREG/DCNT	time	f	case 2 – with DVFS CREG/DCNT	time	f
①	100/100	0	1.00	100/100	0	1.00	100/100	0	1.00
②	75/75	25	1.00	75/90	10	1.00	75/90	10	0.83
③	65/65	35	1.00	65/85	15	1.00	65/84	16	0.77
④	35/35	65	1.00	35/75	25	1.00	35/71	29	0.49
⑤	30/30	70	1.00	30/74	26	1.00	30/69	31	0.43
⑥	0/0	100	-	0/44	56	-	0/0	100	-

Table I shows three execution scenarios. The values in cell are the values of the credit counter and the deadline counter respectively. The assumption is that subtraction of credits is done instantaneously when the node is reached. The first column represents the worst-case execution scenario. When the sequence follows the critical path A-B-C-D-E, there is no slack. If execution follows the non-critical path A-F-G-D-E and each segment executes to its WCET, there is slack of 30. The reason is that the *while* loop does not execute because the entering condition and that *if* statement follows the *else* branch. The third scenario illustrates an actual-case execution when the execution of a segment lasts less than the WCET, because of the input data variation. The sequence is A-B-G-D-E, where the I/O in segment A takes only 40% of its WCET, and *while* loop executes 5 of maximal 10 iterations and the I/O operation in segment D uses 20% of its WCET. At the end, the total accumulated slack is 44. The reason for the shorter execution time of segment A can, for example, be cache misses, predicted in its WCET, but never happened.

We will use the last scenario of the previous example to illustrate how detected slack can be used for power reduction with DVFS. For purposes of this example we will assume ideal DVFS mechanism, i.e. arbitrary level of voltage and frequency can be used within the operating range and that transitions between the levels are instantaneous. Table II gives two possible variants for PM reaction based on the same actual case. For easier comparison, the table includes WCET scenario, and two cases of the same ACET scenario, with and without slack exploitation. All cases are given with the values of CREG and DCNT, timeline and determined frequency. In case 1, DVFS is not enabled so the IP runs always with maximal frequency and in case 2 shows how frequency is lowered to the ratio of CREG and DCNT. As a consequence, execution times of the same segments in case 2 are longer

than in case 1, due to the lower operating frequency. The execution time of the segments are longer than corresponding local deadlines, but the global deadline for the task will be met. This way the slack is not necessarily exploited immediately, but over the remained task execution time. It also prevents the system of making transitions very often which can be contra effective. It is clear that the last segment (E) will use all the generated slack if it runs in its WCET. Otherwise, only the slack of the last segment will remain at the completion of task, but scaled with the operating frequency. We expect that in real-life use-cases of this method, the amount of slack and the power reduction will be much bigger.

VII. Discussion

Through these two simple examples we have shown a simple variant of the hardware and software solution that we propose. It uses the compiler inserted credits as a notion of progress mechanism and based on that hardware makes the decisions how to scale frequency and voltage.

The described approach has as a very valuable feature that it is performance neutral. If proposed solution is implemented as specified, the whole tile always execute in WCET, for almost every actual case. Although, performance can be better on average, this approach still satisfies the specification requirements and consume less power, if not close to optimal.

Another benefit of this approach is the local slack exploitation: all tiles are power-managed independently. It is simple, and there is no need for communication between power management modules. It is easy to combine this local approach with the system-wide slack reclamation methods.

The credit system can be modified to include some events of interest, like cache misses, exceptional interrupts etc. This events can be detected, converted into credits and added to CREG. Another variant

is instead to count deadline always with the critical path, to include some relaxed variant of WCET with optional segments. In that case, when some additional segment has to be executed, credits are added to CREG on the start and then subtracted at the end of the segment.

Another extension of the credit system can be cumulative deadline approach, e.g. if there are buffers with input or output data for the task. The DCNT and CREG can contain the values extended for over multiple task iterations.

The credit system granularity has a big range. Credits may represents workload in cycle count, number of instructions, number of segment iterations up to or user-related metric. Off course, the more fine grain approach is, the more aggressive power management will be, with bigger power reduction.

VIII. Conclusion and Future work

We have presented in this paper the motivation and scope of our research. Our main goal is to provide the combined hardware and software solution for power management of the heterogeneous MPSoC. We want to obtain this goal by exploiting dynamically generated slack using DVFS methods and gating the clock and power. Example has shown how the credit based system can detect and use the slack locally. This is achieved by the collaboration of tools and proposed hardware support for power management.

As the part of the future work, we want to investigate different variants of such credit system with different types of the system resources. We also want to extend this solution for the whole system, as well as couple it with the QoS management. The intention is not to cause any QoS degradation by actions of power management. The hardware support will also be closely coupled with QoS management and promptly react accordingly to its changes.

References

[1] N. AbouGhazaleh, D. Mossé, B. Childers, and R. Melhem. Collaborative operating system and compiler power management for real-time applications. *ACM Transactions on Embedded Computing Systems (TECS)*, 5(1):82–115, 2006.

[2] A. Azevedo, I. Issenin, R. Cornea, R. Gupta, N. Dutt, A. Veidenbaum, and A. Nicolau. Profile-based dynamic voltage scheduling using program checkpoints. *Design, Automation and Test in Europe Conference and Exhibition, 2002. Proceedings*, pages 168–175, 2002.

[3] S. Gheorghita, T. Basten, and H. Corporaal. Intra-task scenario-aware voltage scheduling. *Proceedings of the 2005 international conference on Compilers, architectures and synthesis for embedded systems*, pages 177–184, 2005.

[4] S. Gheorghita, S. Stuijk, T. Basten, and H. Corporaal. Automatic scenario detection for improved WCET estimation. *Proceedings of the 42nd annual conference on Design automation*, pages 101–104, 2005.

[5] K. Goossens, J. Dielissen, and A. Rădulescu. The Æthereal network on chip: Concepts, architectures, and implementations. *IEEE Design and Test of Computers*, 22(5):414–421, Sept-Oct 2005.

[6] A. Hansson, M. Coenen, and K. Goossens. Undisrupted quality-of-service during reconfiguration of multiple applications in networks on chip. *Proceedings of the conference on Design, automation and test in Europe*, pages 954–959, 2007.

[7] P. Heysters. *Coarse-grained Reconfigurable Processors: Flexibility Meets Efficiency*. Centre for Telematics and Information Technology, 2004.

[8] S. Lee, K. Choi, and S. Yoo. An intra-task dynamic voltage scaling method for SoC design with hierarchical FSM and synchronous dataflow model. *Proceedings of the 2002 international symposium on Low power electronics and design*, pages 84–87, 2002.

[9] M. Meijer, J. Pineda de Gyvez, and R. Otten. On-chip digital power supply control for system-on-chip applications. In *ISLPED '05: Proceedings of the 2005 international symposium on Low power electronics and design*, pages 311–314, New York, 2005. ACM Press.