Summary:

The paper proposes an approach called *Parcae*, providing a generally applicable automatic system for platform-wide dynamic tuning.
Despite other parallelization approaches, which merely try to convert sequential programs into parallel ones through e.g. speculation, Parcae combines multiple parallelization techniques with an adaptive system to select the best one at runtime. Moreover, Parcae takes not only the degree of parallelization of a single program into account, instead it provides the possibility of a platform-wide optimization of multiple parallel programs simultaneously, while taking the usage of system resources itself into account too.
The Parcae system itself is composed by two different components:

1. The Parcae compiler, *Nona*, provides the ability of automatic extraction of different types of parallelizable regions out of a given input program. The prototype implementation is capable of extracting DOANY and PS-DSWP style parallelism out of loop nests.
   Additionally, the compiler inserts code to efficiently pause, reconfigure and resume the execution of the parallelized code, which is required in order to perform a run-time adaption of a given task in the program.
   This includes the insertion of profiling hooks for the run-time system in order to monitor the performance of the given application.

2. The Parcae run-time system, consisting out of the *Decima* monitor and the *Morta* executor.
   Since the run-time system needs to know how "well" a given instance of a parallelized version performs with respect to the current execution parameters, there has to be a way to monitor the "throughput" of this instance. This is accomplished by the Decima monitor system via the already mentioned performance hooks inserted by Nona. Throughput is expressed in the number of iterations, which have been completed during a certain amount of time. Additionally, the time the instance spends waiting is measured, enabling the runtime system to detect bottlenecks e.g. for communication in the PS-DSWP system.
   The adaption itself is finally performed by the Morta executor, which selects appropriate different parallelized versions of the region together with run-time parameters such as number of concurrent parallel tasks, to find an optimal degree of parallelism *(DoP)* by maximizing the overall throughput.
   Together with the measures provided by Decima and the inserted hooks by Nona, Morta is able to perform an continuous adaption of all monitored programs in the system. This includes the "hot" modification of execution parameters during runtime up to switching the parallelized version on each iteration of a parallelized loop.

Issues / Open Questions:
During the extraction of the DOANY parallelism in NONA only commutativity relaxations are mentioned in order to ignore certain loop-carried dependencies and insert appropriate synchronization operations for these. However, I'm unsure if this is sufficient, since for arbitrary reordering not only commutativity but associativity is required too.

Summary A2

This week we discuss a paper about "Parcae" which is a system providing flexible parallel execution of programs. Because many parallel implementations of many programs only work optimal for certain areas of the input space and for certain system configurations, it is hard for software developers to find algorithms that perform well outside this area, too. Parcae tries to solve this problem by dynamically tuning the program for the current system configuration, because at execution-time the so called "run-time factors" like the number of available cores, the memory bandwidth and the system workload are known.
In order to generate flexible parallel code Nona, the compiler that is provided by Parcae, extracts several parallelism types from program regions, it inserts code that enables the program to pause, reconfigure and resume itself at run-time and it puts profiling hooks into the code to allow monitoring the performance. During compilation Nona first looks for parallelism in the program dependence graph of the outermost loop nest. Then several transform are applied to the PDG resulting in code packages called tasks. In their current implementation the authors concentrate only on DOANY- and PS-DSWP-parallelization. In a third step Nona modifies the control flow so that the tasks can be managed by the runtime-system.
During execution when a parallel region is reached the run-time system called Morta starts with a sequential execution to establish a baseline performance as a reference value. After ten iterations it starts reconfiguring the current configuration by launching a parallel scheme. Then it tunes the DoP (degree of parallelism, number of threads for each parallel task) and resumes the execution regularly until the best configuration is found. As soon as the environment changes it restarts this procedure to adapt the program to the new system configuration.

Questions / Opinion:

1. Why do the authors highlight the minimization of energy consumption so often. In my opinion energy efficiency is a nice side effect but it cannot be the main goal of such an approach.

2. What is meant by the "hottest" outermost loop nest in 3.1?

3. What are the commutativity annotations the authors talk about in 3.1

1 Summary

In this paper, the authors present Parcae, a system for flexible parallel execution. Parcae aims at reducing overall execution time while also lowering energy consumption. Parcae consists of three submodules: the Nona compiler, that generates *flexible parallel programs* and adds profiling hooks for the Decima run-time system. Decima monitors program performance and system events. The third part is Morta, it executes, terminates, and replaces tasks.
The Nona compiler parallelizes the sequential baseline version with different techniques, and extracts *tasks* for every parallel region. It therefore uses the Multithreaded Code Generation algorithm, and adapts its output to flexible execution.
Morta uses Decima to monitor the processes and to find an optimal configuration by repeatedly searching for a better configuration. Therefore, Morta replaces running configurations with new ones, monitors them with Decima, and compares the to a baseline.
The evaluation sections shows promising results with up to 42% speedups and 84% energy saving.

2 Questions & Opinions

This work combines static parallelization, monitoring, and dynamic code replacement in a neat way that allows to target different hardware successfully.

Parcae: A System for Flexible Parallel Execution

1 Summary

This paper presents a system consisting of both compile time and run time components that auto-tune the program for parallel execution while achieving maximum speed up and least energy consumption. This system determines at the run time the parallel configuration for the program to execute . System has a compiler, performance monitor, and a component that controls the execution of parallel components. Authors observe that for any program there is an optimal number of cores to obtain maximum speed-up. Energy can be saved by switching off the rest of the cores.

The Nona compiler builds the PDG of the program, and tries to relax some dependencies. Then it uses MTCG algorithm to create initial parallel transformations called tasks. It does so by building CFG, and inserting communication and synchronization code into basic blocks. In the next step, Nona converts these tasks such that number of threads needed for each task is determined at runtime. Nona inserts code for the following things. Firstly, after each iteration control is yielded to Morta runtime system which decides whether task should be paused. Register and stack variables are restored and saved before and after each iteration respectively.

Morta runtime system is used to synchronize the tasks. Whenever Morta decides to change the parallel configuration, it sends pause signal to master task which in turn sends it to other tasks. It also sends resume signal to tasks after setting the new configuration.

Decima runtime system is used to monitor performance of the system. Different parallel configurations are tried and Decima makes note of execution time of each such configuration to decide the best. Initially sequential scheme is used for certain number of iterations before switching to an initial parallel scheme. Different degree of parallelism is tried by gradually increasing or decreasing DoP. Execution time needed for each iteration in a certain configuration is noted down to decide the best configuration.

2 Questions

1. Time monitoring is responsibility of which component? Section 3.4 says Decima, while section 5 says it is done by Morta.

Parcae: A System for Flexible Parallel Execution

Summary

The *Parcae* system consists of several components that altogether allow to have a resource saving and flexible parallelized compilation of a program that adopts to the current state of the system it runs on (available cores, memory bandwidth and so on).

The first component is the *Nona* compiler. It compiles a (sequentially given) program by applying different parallelization schemes for loops and augments the generated code (when still represented as SSA graph) with additional code that is needed later for the run-time components to work. Since the code shall be dynamically replaceable between several iterations of parallelized loops, these additions include code that from cross-iteration dependencies from thread-private context (local heap and registers) to the heap as well as some hooks that enable online monitoring of the code performance.

The other two components are the performance monitor *Decima* and the run-time environment *Morta*. The main idea of performance monitoring is trying several implementations for a very short period of time in the given resources and stick with the version of optimal *degree of parallelism* (based on the measured values) as long as the available resources do not change. If they do, this experimental phase restarts. The online replacement of code versions is done by a hierarchical signal passing mechanism which allows to pause the current execution of a (possibly parallelized) loop after the current iteration, to switch to the implementation determined optimal (in terms of execution time and resource consumption) and resume the execution.

The system can also be extended to multiple programs on one hardware and coexist with the execution of other programs, where the relation between operation system and the presented run-time system is not perfectly clear to me.

Open Questions

- What are *hottest outermost loop nests* as well as *min, max* and *sum* reductions?
- What does the `rdtsc` instruction?
- The system is obviously conceptually extendable to more parallelization schemes, but how does that affect the overhead of the system?

Summary Parcae

The paper introduces Parcae, a runtime system which runs adaptable, parallelized programs. Its defining feature is that it monitors the executed programs, and can recalibrate their parallelism scheme and the number of resources they use to obtain higher throughput. A further interesting characteristic is that as a secondary goal it optimizes the schedule to minimize power usage.

The Parcae system consists of several parts:The Nona compiler is able to find inherent parallelism in programs (by analyzing the Program Dependence Graph). It is able to recoginze min, max and sum reductions. It also applies two parallelizing transformations: DOANY and the previously seen PS-DSWP. By keeping multiple versions of the code (sequential and the aforementioned parallel variants), the system can dynamically switch between them to achive the best performance. To enable those switches, the compiler adds further code: A task (body of a loop) yields to the runtime, and asks whether it should continue running (possibly on a different thread), or pause. To support migration, special code to copy values from registers carrying cross-iteration dependencies is inserted. A further part of the the system is the Morta run time system, which determines which verrsion of the code generated by Nona is used. In order to do this, it first executes the sequential for some time, to obtain a baseline value. Afterwards, the system tries several combinations of parallelization scheme and degree. Once every scheme has been tried, the system chooses the one with the best measured throughput and keeps on using it; until Decima, the monitoring system, detects a change of the workload, or of the available resources. Monitoring is partly done using hardware TSCs.

Noteworthy is the approach the authors used to find the best degree of parallelism: They use a finite difference gradient. Unfortunately, they don't really elaborate on why they choose this method. They mention however that their approach has one drawback, namely that it might find only a local optimum.

In the end, the authors evaluate their approach on 2 systems, showing that they can achieve speepup compared to the sequential verion. They further argue that their system can not only be used for a single program, but for multiple ones running in parallel, and show that in this case, their system outperforms the Linux scheduler, achieving better throughput.

Open Questions

- What exactly is DOANY parallelism?
- How can one apply PS-DSWP without the synchronization array?
- Why is the assumption that there is only one minimum justified (in many cases)?
- Why did they choose to use a forward difference for the finite difference?

Summary A2

This week's paper presents a system called "Parcae", a compiler and run-time system to automatically parallelize during compile-time and to adapt execution depending on the current state of the underlying system.

The paper divides its system into 3 parts: the programmer, the Nona compiler and the run-time system. The programmer is advised to insert commutativity annotations into his sequential code which the Nona compiler can use while building the PDG. The Nona compiler itself performs 3 steps: PDG construction, parallelization and flexible code generation.

The system uses 2 parallelization schemes: DOANY and PS-DSWP, applying the one that fits best.

For flexible code generation the Nona compiler simply modifies the code generated by MTCG. MTCG only generates code for a fixed amount of threads and is therefore unsuitable for adaptive execution. Nona applies 5 changes to the generated code to make it applicable for the wanted scenario.

The run-time system monitors the execution and calibrates new configurations with different levels of parallelism. This can be divided into 4 steps: initialization, calibration, optimization and monitoring of the configuration. The baseline that it starts with is always the sequential version. If a parallel configuration turns out to be faster than the baseline, it becomes the new baseline.

When executing, processes running a program in parallel need to communicate information about their iteration to each other. A process will send one of 3 signals when an iteration is finished: pause – which means that the process is currently paused until it receives the signal to continue; iterating – which means that this thread finished an iteration; and iteration-end – meaning that the thread just finished the last iteration of the loop. The pause signals may come from other processes or the run-time system itself.

Open questions:

1. The third change Nona performs to modify MTCG code (page 136, 3.3, bottom): what does it mean and why is it needed?
2. Why do processes need the ability to pause and continue execution?

Parcae: A System for Flexible Parallel Execution

1 Summary

The paper proposes the Parcae system for automatic compiler parallelization and runtime optimization. The system is composed of three components:

- A compiler that extracts parallelism called Nona. Nona will identify potential parallelizable loops in the CFG, it will then try to eliminate some data-dependences using privatization or recognition of reduction patterns such as min,max,sum etc. Some dependences may inhibit concurrent execution but not unordered execution, Nona will allow the programmer to annotate dependences to be commutative to handle these cases. After resolving or relaxing as much data-dependeces as possible Noma will produce multiple output programs: A normal sequential version and multiple parallel version that are obtained by using standard loop nested parallelization techniques (in their example PS-DSWP and DOANY, they claim others can be adapted).
- A runtime calibration and execution tool called Morta. Morta will profile the programs generated by Nona for a set amount of iterations and compare their performance measured by iteration throughput. To get comparable values Morta will execute each program on its own for a fixed set of iterations (defaults to 10) and with different grades of parallelism. After an iteration finishes every thread waits for Morta to assign the next task or end the execution. The configuration with best throughput will then be executed. Morta is also able to reconfigure at a later point in time if a change in workload is detected or system resources change.
- A system monitoring utility called Decima. The task of Decima is to passively monitor the execution of the parallelized program and available system resources. If a change in workload is detected Decima will notify Morta to schedule a reconfiguration.

2 Questions

- When Decima detects a change in workload and schedules a reconfiguration through Morta, how are the states of tasks that have finished preserved before reconfiguration?