Summary A1

This week we discuss a paper dealing with "High-level optimization via automated Statistical Modeling". With this paper the authors present an automatic algorithm and data layout selection approach using models built from profiling information to pick the best suiting one for the used hardware. This technique is very useful for implementing libraries when the the underlying hardware varies because the programmer can implement different versions of the algorithms and the runtime-system chooses the best one at runtime.
To make this selection possible each implementation comes with a model that predicts the estimated execution time on the target hardware in dependent on a set of input parameters like the problem size, for example. These models are automatically generated by the so called "auto-calibration-toolkit" in order to ensure robustness and accuracy. When porting an algorithm to a new platform it is not necessary to rewrite the algorithm any more, the only necessary thing is to port the runtime system which is straightforward and not so cost expensive as the authors mention.
To evaluate their approach they use four different platforms which have all different architectural characteristics. They mainly differ in the speed of the communication between their nodes and as a result have different performances on the implementations. First they show a benchmark on stencil computations and compare four different data layouts: "uniprocessor", "square blocks", "vertical stripes" and "shared memory". The results impressively show that on the one hand certain ranges of the problem size are especially suitable for a certain layout and on the other hand that in 99% the prediction chooses the right layout variant. Even when the predictor fails it chooses an implementation performing nearly optimal. Besides the stencil computations the authors also provide a benchmark on sorting where the predictor has to choose between several radix-sort implementations for smaller problems and a sample-sort implementation for bigger problems. The positive result could be confirmed here, too.

Questions / Opinion:

1. What is linear regression and what is it used for in this approach

2. Since personal computers are quite similar in their architecture, can this approach also provide benefits for them or is it only useful for supercomputers?

1 Summary

In this paper, the author presents an high-level optimizer for automatic code selection and parameter optimization, based on statistical modeling.
The goal is to select out of a set of implementations the one that performs best on a specified target hardware. To do so, models to predict the performance are automatically extracted from the concrete implementations using the auto-calibration toolkit.
To evaluate the approach, the author considers two problems: stencil computation and sorting. For stencil computation, there are four concrete implementations, and for sorting the author compares a parallel radix sort and parallel sample sort.
Overall, the statistical models show an accuracy of over 99% choosing the optimal implementation.
In addition to selecting the best concrete implementation, the author proposes a technique to optimize existing implementations by tuning the input parameters. Therefore, he combines statistical models with numerical root finding.
For stencil computation, the author describes a model that relates grid points per node to the number of directions of a node. Nodes with fewer directions will be assigned more grid points to ensure load balancing. For radix sort, parameter optimization can be used to find the *overall* optimal radix. Here, all performed automatic optimizations have been optimal.
The author explicitly states, that his high-level optimization framework can not only be used statically (at compile time), but also be integrated to software ad evaluated dynamically. While dynamic algorithm selection highly impacts code size, this can be neglected for automatic parameter optimization.

2 Questions

Although the results are quite impressive, I do not see how this can be used for automatic parallelization. The concrete implementations must already be parallelized; the framework only chooses an optimal one. I hope this will be elaborated in the meeting :)

High level optimization via automated statistical modeling

1. Summary

Portable low-level code portion of a program may not be the best optimized for a given platform. This is mainly because of many possible data layouts and algorithms. This paper discusses how statistical modeling can be used to choose the best possible optimization for a particular platform. There is an auto-calibration toolkit that generates the statistical model for any program using the profiling information for the platform. Implementation with the minimum predicted execution time for the particular platform is chosen as winner. Author uses four target platforms to test his test cases.

First test case is the stencil computations that are used to solve partial differential equations iteratively. Problem space is divided into a virtual grid. Four types of data layouts on the grid are discussed. Namely uniprocessor layout, square grids, vertical layout, and shared memory. Each of these layouts have different communication overheads. For smaller inputs, uniprocessor layout wins. Square grid layout performs well when height and width of the grid is almost identical and so on. So depending on the virtual grid size toolkit can choose appropriate data layout.

Sample sort and radix sort algorithm with different radices is compared next. Sample sort avoids rearrangement of data for each digit by using boundary keys so that keys are moved only within a boundary. But it has large memory requirements. So small-radix radix sort works better for smaller inputs. But for large inputs sample sort beats radix sort.

Next, author discusses about using root finding with parameter range of interest to evaluate the statistical models. Binary search can be used for efficient root finding. Stencil programs are optimized by load balancing.

2. Open Questions

1. What is the strided block transfers required for horizontal data layouts discussed in section 3.1?

High-Level Optimization via Automated Statistical Modeling

Summary

This paper describes a system to implement *high-level libraries* which adopt their own behavior depending on the parameters they are called with as well as the platform they are executed on. This is done either at run-time or during compilation.

The main idea of this approach is – instead of giving a single implementation of an algorithm – to put several implementations for one functionality behind an *algorithm selector* which then makes a choice for the given problem based on three kinds of information: parameters of the problem (e.g. problem size), the run-time environment (cost functions etc.) and *models* of the different algorithms available. Those models are the core of the system. They are basically approximating cost functions for each implementation. As it occurs, they need to be partially given by the programmer in the form of several terms (describing a particular dependence on a parameter) which are then weighted (or ruled out) by some refinement process called *auto-calibration toolkit* that uses statistical means such as linear regression for that. The paper emphasizes that this approach is very precise in the prediction of the real run-time of the algorithms they investigate and even more in the choice of the optimal algorithm based upon that.

They apply their system on two use-cases, once stencil computations, which seem to be common in large-scale simulations, and array sorting. Both problems share the characteristic that they are easily dividable but that the performance on distributed multi-CPU systems depends on the way one partitions the problem into (almost) distinct sub-problems. They (apparently successfully) use their system to cope with that.

Another nice benefit of that approach is that some implementations of an algorithm can be partially implemented with respect to the input range as long as there are backup implementations for the cases that are not covered.

Open Questions

- Although it doesn't seem to be that relevant for the paper by itself: What are stencil computations exactly?
- I wonder how universal the presented modeling system is, given the repeated use of the word "linear". Is that because we are just comparing the linear gap between algorithms of similar complexity? Or what exactly are the underlying statistical concepts capable of?

Summary A1

This week's paper, called "High-level Optimization via Automated Statistical Modelling", presents an approach to tackle the problems algorithm libraries face when dealing with different hardware. The problem is that certain implementations might not run or run slowly on different hardware than they were designed for. As porting implementations can be a really tedious task, they present an approach to make this obsolete.

The idea behind the approach is to have multiple versions of certain algorithms at hand which a runtime system can select from to achieve the highest possible speed. This selection is done based on the input and the underlying machine.

For each version of an algorithm the runtime on the target machine is predicted with respect to certain input parameters. The models which predict the runtime are automatically generated from input ranges and terms provided by the programmer.

The authors claim that with their approach porting a library to another platform is very easy. Instead of porting the complete library, only the runtime system to select the optimal version needs to be ported.

High-Level Optimizations via Automated Statistical Modeling

1 Summary

The paper proposes a technique to optimize libraries by choosing different algorithm implementations depending on actual input and platform-profiling information.

Using their system involves some extra work for programmers to include it in their libraries. When designing the library the programmer is required to first provide a specification for the problem that will be optimized, the paper does not describe how this is represented. After defining a specification for the problem the next step is to provide multiple (at least one) implementation for the problem. Different implementations do not have to handle all possible inputs, the programmer can specify input restrictions for each different implementation to achieve this behavior. The last requirement is for the programmer to find terms which can be used to express the predicted performance of each implementation.

When calibrating the library for a specific platform the system will perform linear regression (they don't go into detail) to find coefficients for each programmer specified term on data acquired by execution on the platform. The system will ignore terms that are statistically irrelevant, unhandled inputs will lead to a performance prediction of infinity. These coefficients form the models that predict the performance of each implementation, they will be used to select the statistically best implementation for each input.

2 Questions

- Are they ensuring that there is an implementation to choose for every input? Does it make sense to do so?
- Does the calibration script output code that has to be compiled or does is produce some abstract model description that can be read by the application?
- How does the programmer find good terms and what properties should a good term have?
- What parameters does their profiling use to derive models? Is this fixed or can the set of parameters be extended when calibrating?
- How are terms decided to be statistically irrelevant?