Simulator-based Exploration of the Memory Hierarchy for Data Dominated Applications

Jeroen van der Vegt <ajvdvegt@123mail.org>
Erik Brockmeyer <brockmey@imec.be>
Georgy Gaydadjiev <G.N.Gaydadjiev@ewi.tudelft.nl>

Abstract—**Battery capacity cannot keep pace with increasing energy requirements for portable electronic devices. These devices are often equipped with multimedia applications, which require an increasing amount of energy as the image quality is continuously improved. It has been indicated that video compression causes the biggest part of the total energy consumption, hence it is a potential candidate for tremendous energy savings. Energy can, for example, be saved by applying multiple, low power, low speed processors, and by efficiently using the memory subsystem. To exploit the memory hierarchy in a multiprocessor environment, the developer needs tools that automatically transform his program to fit the targeted hardware This paper describes a simulator which is used to evaluate the modification of such code-rewriting tools for multiprocessor platforms. The simulator focuses on the memory subsystem and assumes that memory access timing is dominating the total execution time. Therefore, time spend on computations is completely ignored. The simulator estimates execution time and energy usage by automatic annotation of the user-program's source code. After annotation, the source code is compiled and a fast, high-level simulation is performed.**

An MPEG4 video encoder is used as a case study for both the simulator and the code-rewriting tools. This encoder was adapted to run on a parallel platform, and it is used to illustrate the modifications required to make use of the simulator. The paper also illustrates how the simulator can be used to find potential performance bottlenecks in any parallelized algorithm.

It is shown that using six processors instead of a single processor improves the performance by approximately a factor three. Simulation results are analyzed to explain the difference with the theoretical factor of six.

Key-words: Multiprocessor, memory, MPEG, simulator.

I. Introduction

Several years ago, the Data Transfer and Storage Exploration methodology [5] (or DTSE for short) development was started at IMEC. This technique focuses on data usage, and tries to reduce power consumption of a computer platform's memory subsystem. In addition, it increases performance of data dominated applications, such as multimedia and telecommunication algorithms. The methodology works at the source code level, which means original code is modified in order to implement the optimizations. It can therefore be seen as an extra pre-compiler stage. The DTSE methodology will be discussed in more detail in Section III. Previously, DTSE has mainly targeted environments with a single thread of control for custom design. For the future, IMEC envisions a trend towards multiprocessor architectures. The current DTSE toolkit will therefore be extended for platforms with a variable number of processors, which have different constraints with respect to the memory hierarchy than single processor platforms. The toolkit will specifically focus on the assignment of data in the memory hierarchy. To steer future research within IMEC, more information is needed about the problems arising in such multiprocessor environments. As a first step for this multiprocessor research, the timing versus energy trade-offs for multiprocessor environments when changing the memory hierarchy are explored in this report. In order to perform such exploration, a simulator is a very useful tool. A dedicated simulator has been developed to exactly fit the exploration needs, and to closely integrate with existing tools. For easier reference throughout this report, the simulator will be referred to as MuPSi which is short for **Mu**lti**P**rocessor **Si**mulator.

II. Target architecture and simulator

An important reason to build a simulator is that it is very hard to predict what will happen when scarce resources run out. Models to describe this are hard (if not impossible) to develop, but insight in the potential problems could result in better performance. An example of a scarce resource that might run out is a data bus, contained by virtually all computer platforms for data transport between different components. When several components try to use the bus simultaneously, bus congestion occurs. When no special measures are taken, this is more likely to happen on a multiprocessor platform than on a single processor platform. As the simulator is targeted to multi-processors platforms, bus simulation is an important aspect of MuPSi˙To mimic a real platform with any reasonable accuracy, *only* taking the bus into account would not be sufficient. Therefore a commonly used platform layout was examined, and all key elements identified. All these elements were added to MuPSi˙These elements are depicted in Figure 1 as squares, and listed here for convenience. The straight arrows in the figure represent a data line, curved arrows indicate a control line.

Processors: These are the functional units in a platform. As MuPSi focuses on the memory subsystem, the processors are largely disregarded. Separate processors are implemented as POSIX[4] threads,

Data bus: This is used to copy data over between different components of the platform. The main targets of this report are small, low power systems where the load is spread over multiple processors. Therefore a simple shared bus was implemented. There is nothing however that prevents extending the model with other, more complex bus models. Even though the resulting bus model is fairly easy, it *does* take bus conflicts into account, and it is possible to assign priorities to bus transfers.

L1 memory: This is a small and fast memory connected directly to the processors. Traditionally, L1 memory is a data cache memory. In this environment it can be regarded as a software-controlled cache, or 'scratch-pad' memory.

L2 memory: This is larger memory further away from the processors, connected through the data bus. Accessing this memory takes much longer than accessing L1 memory, and it consumes more energy.

DMA/bus controller: This controls block transfers and accesses to the bus. This is assumed to be a separate functional unit, but its energy consumption is not taken into account.

Please note that MuPSi is set up in a flexible way, hence not limited to this example. A real-life (but relative simple) example of such an architecture is for example the BlackFin chip from Analog Devices, as mentioned in [1].

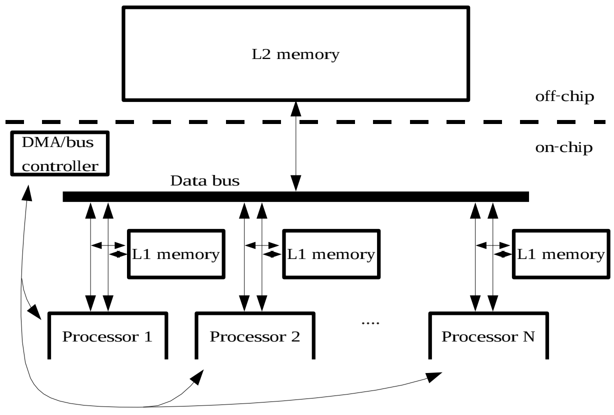

Fig. 1
A GENERAL MULTIPROCESSOR PLATFORM.

For MuPSi there has to be an easy way to judge the relative quality of different mapping alternatives, in terms of power consumption and performance. MuPSi will therefore concentrate on the memory subsystem. In the current implementation, the processor computation time is ignored, assuming performance of the data subsystem is dominant.

In order to decrease development time and increase the maintainability of the source code, a modeling environment language like SystemC [3] was used, called Tipsy [2].

III. ENERGY SAVINGS

A key issue in the DTSE methodology is the notion of data locality. Data locality means that while data is often stored in arrays, a lot of computations involve only a small part of those arrays. Traditional processor caches might not be able to exploit this fully. However, analyzing the program code beforehand and rewriting parts of it, gives an opportunity to reduce the energy consumption. An extra gain of this action is that it can potentially increase performance.

An example of the energy savings using data copies is depicted in Figure 2. In this example, a program accesses array A in a loop, in such a way that every element in the array is accessed 5 times. In the figure, an architecture with two layers of memory is depicted: a large memory where the array A is stored, and a smaller memory where a copy of this array, A', is stored. Accessing the smaller memory costs only a tenth of the energy of what accessing the larger memory will. The total number of accesses to the array is 100%. Without A', the energy costs are also 100%. When the copy A' is introduced, copying the data generates some accesses to A as A' has to be read from it. All accesses to the data are still performed by the processor, which causes the same amount (100%) of accesses to A'. So using copy A' generates more accesses (20 per cent in this example), but the total energy used is less (down to 21 per cent). Arrays like A' are called 'copy candidates'.

Data dominated applications often have multiple layers copy candidates. These can be depicted in so-called 'reuse-trees', one of which is depicted in Figure 3 for an actual MPEG4 codec. In this figure, arrays (with their name in the source code) are shown in red on the top. The arrows at the bottom point to accesses to these arrays in the code. From bottom to top, copy candidates grow bigger and they will have less misses. 'Less misses' means that higher memory levels need to be accessed less often, which is reflected in a so called *reuse factor*. The more accesses to higher memory levels can be avoided, the higher the reuse factor. A copy that would only be read once has a reuse factor of 1, and is actually an ordinary data pre-fetch.

For every access at the bottom there is a single path to

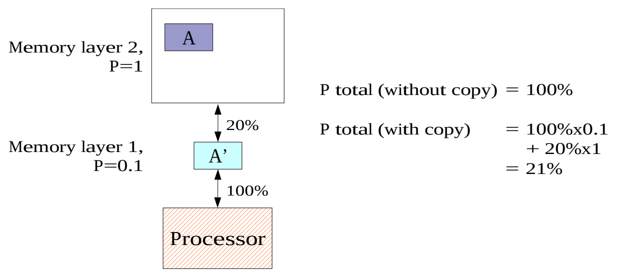

Fig. 2
EXAMPLE OF ENERGY SAVINGS IN A TWO-LAYER MEMORY SYSTEM.
P means 'power used', A and A' are arrays.

the top. For every access in the source code, a copy candidate can be found for every loop the access is nested in. Multiple copy candidates on an access-path therefore indicate that the access is nested in multiple loops. To find the best selection of copy candidates for a particular hardware environment, a tool called MHLA (short for Memory Hierarchy Layer Assignment) was developed.

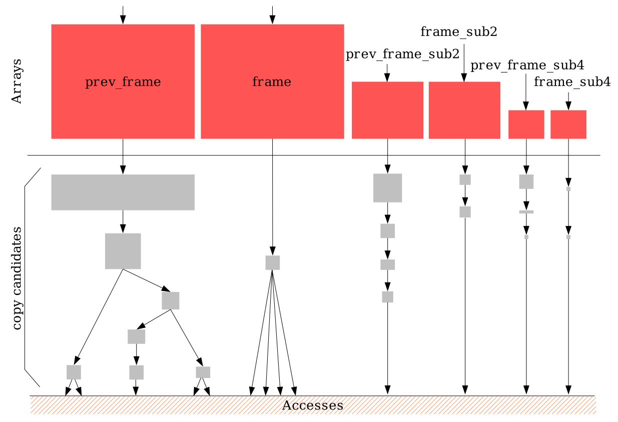

Fig. 3
A REUSE TREE FROM AN MPEG4 CODEC.
All nameless rectangles are copy candidates.

IV. RESULTS

A simulator has been developed which allows for fast and easy exploration over several hardware aspects to minimize the energy consumption of an particular program on a particular hardware platform. To illustrate the potential use of the simulator, a real-life example of a platform optimization has been performed. For this example, an optimal platform for the QSDPCM [**?**] MPEG4 codec was sought. The general layout of the target platform in mind was like the platform in Figure 1, where the number of processors and the L1 sizes had to be optimized. As start of the exploration, the optimal L1 size for a single-processor platform was explored. The results are depicted in Figure 4, and they indicate that using 2.048 bytes L1 memory result in lowest execution time.

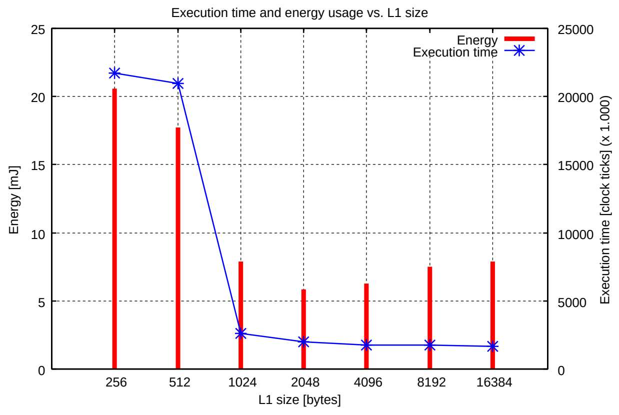

Fig. 4
ENERGY AND TIMING RESULTS FOR NO PARALLELIZATION AND INCREASING L1 SIZES.

Building on these first results, the number of processors was increased while the L1 memory size was kept constant. In order to utilize multiple processors, the algorithm had to be adapted. Intra-frame parallelization was chosen as best results were expected with that parallelization, and this parallelization does not introduce additional first-frame delay. For the parallelization, a frame was split in several rows and column. The number of rows and columns are run-time selectable, thus allowing for numerous different parallel implementations of the algorithm. Restrains in the MPEG algorithm prevented the use of more than nine processors this way, but this turned out to be enough for the exploration. The results are depicted in Figure 5.

This figure indicates using 6 processors results in fastest execution and lowest energy consumption. More detailed analysis of the results show bus congestion to be a major bottleneck. This in turn is caused by sub-optimal parallelization. An clear example of this is indicated in Figure 6. In this figure, the analysis of a platform with seven processors is shown. In this platform, every processor has his own DMA controller, but all DMA controllers share a single bus. For every DMA controller, their line is high when it is active, and low when inactive.

Graphs like this give much insight in algorithmic problems and resulting hardware bottlenecks. For example, the large gaps in the graphs for DMA controllers 5 and 6 turn out to be caused by bad parallelization.

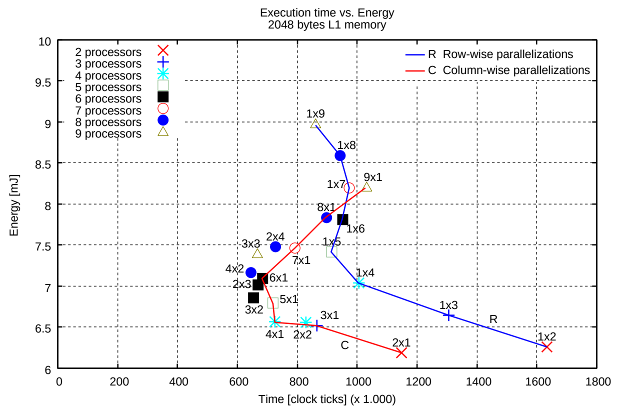

Fig. 5

ENERGY CONSUMPTION VERSUS EXECUTION TIME FOR VARIOUS PARALLELIZATIONS.

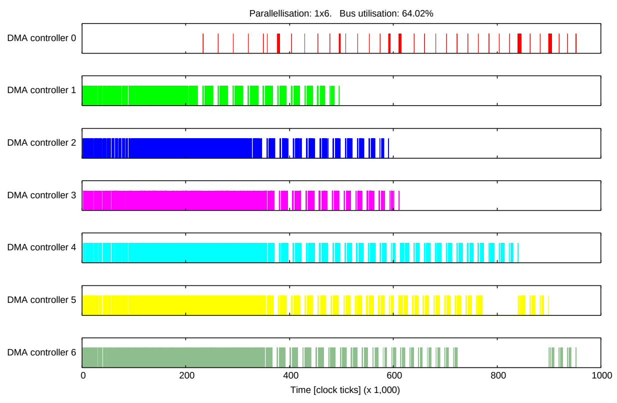

Fig. 6

BUS USAGE OF INDIVIDUAL DMA CONTROLLERS USING A SEVEN-PROCESSORS PARALLELIZATION.

V. CONCLUSIONS

A multiprocessor platform simulator called MuPSi has been developed. It allows for flexible creation of different platforms, and fast exploration of various hardware aspects. It is targeted towards data dominated applications, and has been validated using an MPEG4 video coder called QSDPCM. This exploration has shown some of the problems that arise while parallelizing algorithms, and about the prediction of performance of algorithms. An important goal of the simulator was to allow easy simulation of parallel algorithm execution on various hardware environments, thereby obtaining detailed information about the memory accesses. MuPSi fulfills the targeted requirements by imposing no limitations to the hardware to be simulated, and performing the simulation fast. Based on simulator runs using the non-parallelized version of the algorithm, several parallelized versions of the QSDPCM MPEG4 algorithm were selected to be evaluated using MuPSi. Best performance was obtained using six processors, but results showed a speedup of factor three instead of the theoretical factor six. Results from MuPSi give detailed insight in what problems arise, and can thus be used to create an optimized program for a specific hardware platform.

REFERENCES

[1] Blackfin, *Blackfin bf561 data sheet*, www.analog.com/UploadedFiles/Data_Sheets/696848ADSP-BF561_HiSpeed_pra.pdf .

[2] Johan Cockx, *Tipsy homepage*, www.imec.be/design/background/tipsy/ .

[3] Thorsten Grotker, Stan Liao, Grant Martin, and Stuart Swan, *System design with systemc*, Kluwer Academic Publishers, 2002.

[4] The Open Group, *Ieee posix 1003.1c standard*, IEEE Standard for Information Technology, 2003, www.unix-systems.org/version3/ieee_std.html .

[5] P.Panda, N.Dutt, A.Nicolau, F.Catthoor, A.Vandecappelle, E.Brockmeyer, C.Kulkarni, and E.De Greef, *Memory organisation and optimizations in application-specific systems*, IEEE Design and Test of Computers, vol. 18, June 2001, pp. 56–69.