Remove: A Computer Architecture Designed for Modern VLSI Technology

Steven Roos Reinoud Lamberts Henk Corporaal

Department of Information Technology and Systems,
Delft University of Technology,
P.O. Box 5031, 2600 GA Delft, The Netherlands
s.roos@its.tudelft.nl r.lamberts@its.tudelft.nl h.corporaal@its.tudelft.nl

Keywords: computer architecture, VLIW, VLSI, multimedia, instruction level parallelism, instruction scheduling

Abstract

Reduced feature sizes offer the possibility to build more powerful computer architectures. Unfortunately the rapid developments in VLSI technology which allow those improvements introduce another problem. As wire delays do not scale with smaller feature sizes, communication will become a major bottleneck in future processors. Current computer architectures are not suited to handle this problem.

In this paper we introduce the REMOVE *architecture, which is a computer architecture designed for future process technologies. It is built around a large number of arithmetic units organized in a very regular linear array. These units provide the huge computing power required for computational intensive applications like multimedia programs. Each unit writes its results onto its own, single-source driven bus, which can be only read by a limited number of its closest neighbors. This interconnection network keeps power consumption and communication delays to nearby units as short as possible.*

Remove is programmed like a VLIW. For the compiler back-end we propose a dedicated scheduling algorithm that integrates time slot assignment, unit assignment and even partially includes register allocation.

1 Introduction

In modern VLSI technology, wire delays are becoming dominant over gate delays [1]. Therefore modern computing systems are limited by communication rather than computation. The challenge of computer architecture design is not to fit tens or even hundreds of functional units on a single chip, but to feed them with instructions and data. The key to exploit the benefits of the VLSI technology is locality. Global communication (e.g. to centralized resources) should be avoided.

In this paper, we propose a new architecture, the *Remove* architecture, which fully exploits the potential of VLSI technology. The core of the architecture consists of a linear array of arithmetic units. Each unit can write its result on a private bus, which can be read by a limited number of units in its neighborhood.

We believe that Remove has four key advantages over traditional architectures:

- The architecture has a very regular structure. This greatly simplifies the hardware design and reduces the time-to-market. The symmetry may also be exploited by the compiler.
- The architecture offers plenty of processing power, especially when compared to the chip area it uses. Remove is therefore an attractive low-cost solution for computationally intensive applications.
- The architecture is suited for dense VLSI technologies where wire delays are dominant, because its interconnection network consists of relatively small, local busses only.
- Very high operating frequencies are possible as the communication latencies are small due to the usage of short, single-source driven busses.

In Section 2 we will analyze the developments in processor design which motivate the Remove architecture. The architecture itself is described in Section 3. In Section 4 we propose a scheduling algorithm, which forms the main part of the back-end of the Remove compiler. Section 5 concludes this paper.

2 Trends in computer architecture

In the last decade we have seen rapid developments in fields related to computer architecture. In the following subsections we will investigate the developments in three areas: VLSI technology, computer architecture design, and applications for computer systems.

2.1 VLSI technology

The area of VLSI technology shows rapid developments. The density of transistors on a chip increases by approximately 50% every year. The speed of the circuits made with these transistors shows a comparable improvement. However, this is mainly due to the decrease in gate delays. Wire delays decrease at a much lower rate [1].

A very illustrative paper that shows this trend is published by Matzke [2]. He investigated which percentage of the processor die can be reached within a certain number of clock cycles. He made the assumption that wire delays and gate delays should match in a well balanced circuit and that one clock cycle can be as long as 25 gate delays. The results are given in Figure 1. In the $0.25\mu m$ process which is used for current state-of-the-art processors, the whole die is reachable in one clock cycle. However, for the next generation ($0.18\mu m$), only about 55 percent of the gates can be reached in one cycle. This number decreases to only 5 percent for $0.06\mu m$. This development will have a strong influence on the design of future computer architectures.

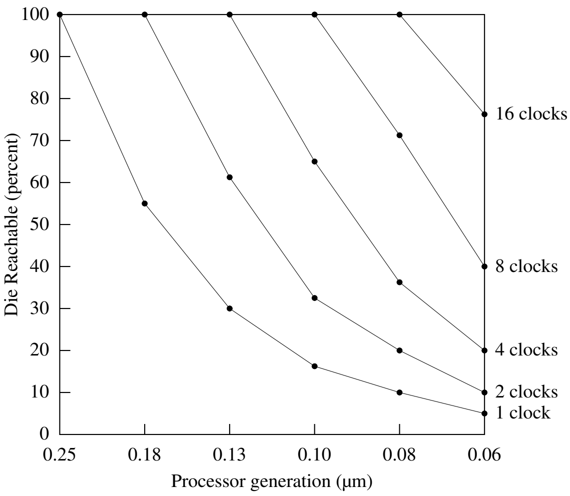

Figure 1: Percentage of the processor die reachable within a certain number of clock cycles for future processor generations. Reproduced from [2].

2.2 Computer architectures

A problem of most current processors is their complexity. Especially the hardware scheduling algorithm in superscalar processors is very complex. This is the reason why there are hardly any dynamically scheduled processors which can issue more than four instructions per cycle. The complexity of the interconnection network is another problem. The ports to the central register file are a serious bottleneck in processors with a large number of functional units.

The complexity of a processor can be reduced by clustering. This also lightens the problem signalled in Section 2.1 that chip-wide communication will be too slow for future generations of processors. A very early clustered VLIW architecture was the Multiflow Trace [3]. Recently this example has been followed by Texas Instruments in their C6x DSP processor [4]. An example of a superscalar processor in which (a light form of) clustering is applied is the DEC Alpha 21264 [5]. In this processor, the register file is duplicated, which halves the number of read ports in each file. In the multicluster architecture proposed by Farkas [6] is a superscalar architecture in which the register file and the functional units are divided over two clusters. If an operation uses or writes operands from the other cluster, inter-cluster copies are dynamically added to the operation.

Some people have even gone further in distributing resources. The multiscalar architecture [7] splits the instruction stream in tasks that are executed on several small processing units. The tasks are assigned by the compiler. Trace architectures [8] restrict their tasks to traces from a trace cache, which are dynamically assigned to the processing units. If the clusters are explicitly programmed as independent processors, a multiprocessor on a chip is created [9].

Though clustering may reduce it, the problem of complexity is not completely removed. The clusters are usually made as large as possible to reduce the inter-cluster communication overhead. In highly clustered processors, the individual clusters may be more complex than complete processors of the same size because each cluster is in fact a complete processor extended with inter-cluster communication facilities.

An additional problem of many clustered architectures is that the old model of centralized resources is still applied for the clusters. This model implies that the number of computational resources is kept small to be able to provide full connectivity between all resources. This however makes them less suited for applications with high parallelism, for which a large processing capacity is more important.

2.3 Applications

Multimedia applications are a growing consumer of computing resources. The prediction is that they will be responsible for the vast majority of the computing cycles consumed in the near future [10]. Des-

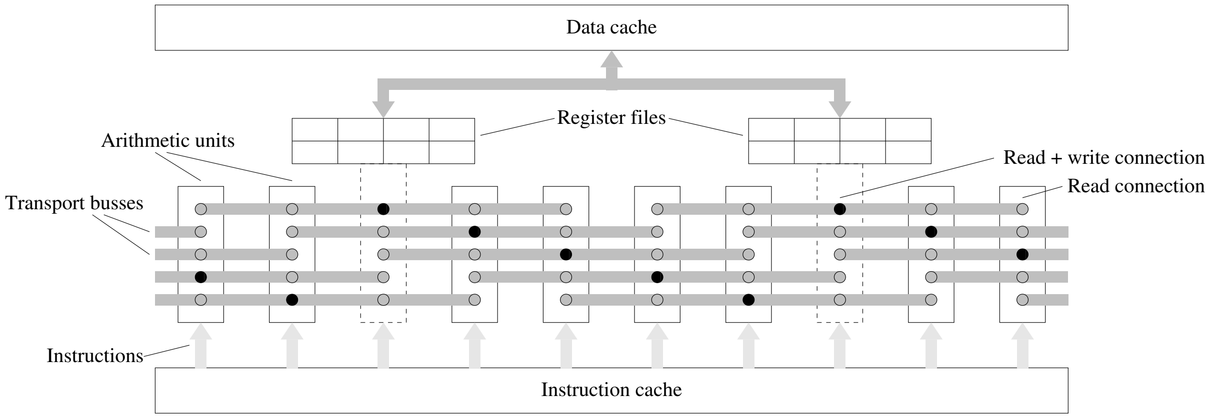

Figure 2: Block diagram of Remove

pite of their diverse nature, media applications generally share three key characteristics.

1. They operate on large streams of low-precision integer data. Image pixel data are generally represented in 8-bit entities, while audio applications usually operate on 16-bit samples.
2. Operations on different streams are largely independent. Therefore large amounts of parallelism are available, and latency may be tolerated.
3. They are computationally intensive. Depending on the application, the number of arithmetic operations per memory reference can be up to one order of magnitude higher than for traditional integer applications. Examples of computationally very intensive applications can be found in the area of 3-D graphics.

Current general purpose processors are usually not suited for these applications because their processing power is too limited. They rely on a low number of centralized resources, where for media applications a large number of resources with restricted connectivity is preferred. Therefore these applications are well matched to the characteristics of modern VLSI technology.

3 Remove Architecture

Remove is a microprocessor which is designed to fit the needs of contemporary VLSI technology. The core of the processor is formed by a large linear array of arithmetic units, as can be seen in the block diagram in Figure 2. Every arithmetic unit writes its results to a single-source bus, which can be read by a few neighbors on both sides.

The regularity of the processor offers advantages for the hardware design. Only one arithmetic unit has to be designed, which can then be replicated many times. This feature largely reduces the design time of the processor.

Apart from an ALU, the arithmetic units in Remove also contain a register which holds the result of the computation performed during the previous cycle. This result can directly be used by the neighboring units, and does not have to go through the register files. This makes Remove fundamentally different from traditional VLIW processors, which contain one centralized register file into which all results are written.[1] The register files shown in Figure 2 are mainly used for the memory and I/O system described in Section 3.2, though can also be used for extra storage capacity.

The power of this processor lies in the structure of the array with arithmetic units. The transport network has a low latency as it only consists of short busses driven by a single source. This allows a very high operating frequency. A dense packing of units is possible because because of the regularity and because short busses won't take much space. Even a low cost processor could contain 20 or more units. Remove is therefore especially suited for applications which require huge numbers of computations on relatively independent data elements. Many media applications fit this description.

3.1 Arithmetic units

An arithmetic unit of Remove consists of an ALU and a result register, as can be seen in Figure 3. It has three inputs: two normal operands and a predicate input. Selectors are used to select the result register from a neighboring unit from which the value is used as an operand for the ALU. The output of the ALU is written into the result register, unless the predicate input evaluates to *false*. In that case the register keeps its old value.

The ALU contains all simple arithmetic and logical operations, like ADD, CMP, bitwise/logical AND, etc, but does not contain separate shift and multiply

[1]In clustered VLIW processors, not the complete processor, but rather each cluster is organized around a centralized register file.

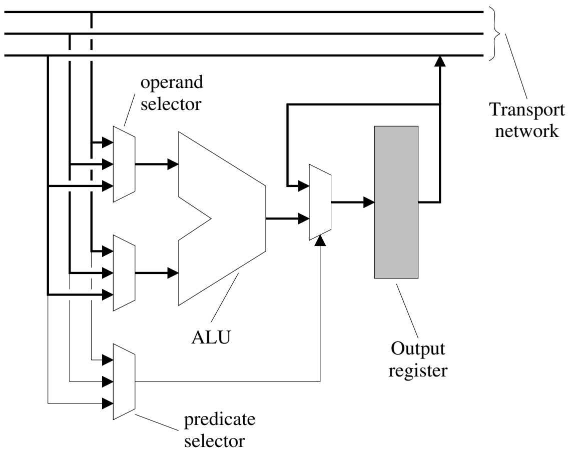

Figure 3: Block diagram of one arithmetic unit in Remove. The support for shift and multiply operations is not shown in this figure.

operations because these functions are too expensive. Instead of a complete shifter and multiplier, each unit implements a limited number of fixed shifts, and part of a multiplier array. Full shifts and multiplies are supported by letting a number of units work together, or by using the same unit for more than one cycle. The exact implementation is still under investigation.

Note that a number of ALUs put behind each other resembles the structure of a multiplier array, with the advantage that the resources for the multiplier can also be used for other goals. Another advantage is that the multiplier can be scaled to the size of the operands. If the operands are known to have only a limited number of bits, less units have to be allocated for the multiplication. However, the exact implementation is still under discussion.

The register in the functional unit is put on the result side of the ALU, and not on the input side. The main reason for this decision is power consumption. If the registers would have been on the input side, the bus to the other units would be driven directly by the output of the ALU. This output is not stable for a long time after the start of a new clock cycle, and causes a huge power dissipation if it is directly used to drive a device with a large capacitance, e.g. a bus to other units.

The placement of the register is however still under discussion. Only one register per arithmetic unit may be too few for a well-balanced machine. With registers on the input instead of the output, code scheduling optimizations are possible. Registers on the input remove the requirement that all operands arrive at the unit at the same cycle. If one of the operands for an operation is available earlier than the others, it can already be moved into an operand register before the operation is started. An operand that is used for several subsequent operations on the same unit can be kept in an operand register.

3.2 Register files and I/O system

The register files in Remove have a dual function. They combine the functions of a normal register file and a load/store unit. They are connected to the surrounding units as a normal arithmetic unit, but they are physically placed above them to provide the necessary bandwidth to the data cache. Within the array all available bandwidth is used for the data busses and the instruction supply.

The I/O system for Remove is designed for a very high bandwidth to the data cache. The bus between the processor core and the data cache is very wide to let memory operations move more than one word from and to the data cache at once. A load will overwrite several registers, just as a full-sized store will write the values of several registers into consecutive memory locations. Traditional (single-word) stores are supported for stores where only a single data word is allowed to be overwritten.

The registers of the register file can also be used for additional storage if the registers inside the arithmetic units are not enough. As most of the operands for the arithmetic units come from the outputs of other units, a high bandwidth to the register file is not needed. The register files in Remove therefore have only one read and one write port.

3.3 Multithreading

For very large configurations, the size of the processor core might become unpractically large. In that case, we can split the processor into several largely independent cores, with a private control flow for each cluster. This is shown in figure 4. Each cluster is a Remove processor for itself, with the exception of the data cache, which is shared. FIFO buffers are responsible for the communication between the clusters.

The clusters operate in a way similar to the multiscalar architecture [7]. The application is divided into several largely independent tasks by the compiler. Each task is assigned to a cluster which executes this task, and communicates its results to its successor. In a multicluster Remove task spawning and communication is completely controlled by the compiler. This is different from the multiscalar architecture, in which these functions are partially done by the hardware.

The advantage of a multithreaded organization is that the execution of control flow instructions does not have to be in order. For example, a task can be one iteration of an outer loop. The second iteration can start execution, while the first is not yet ready. This increases the available parallelism [11], and will im-

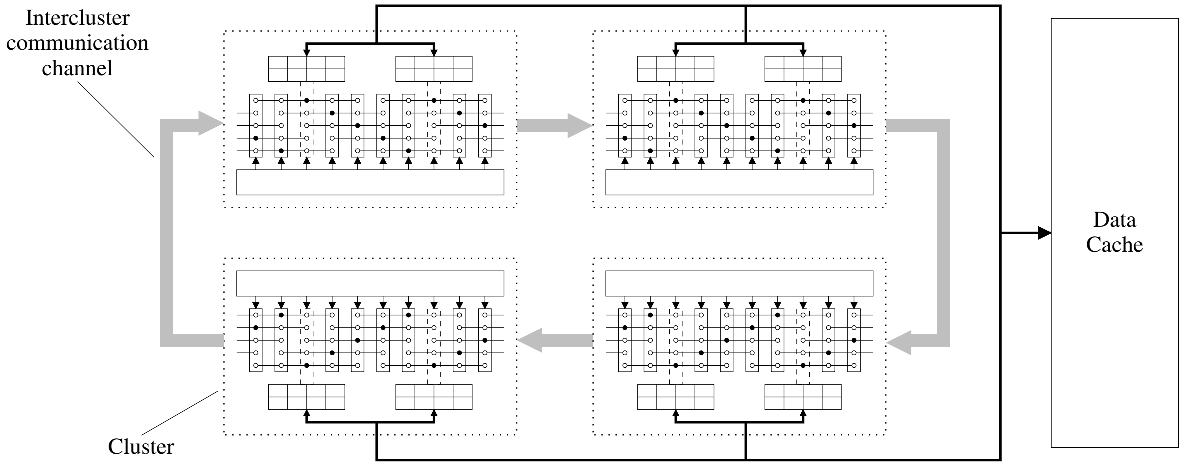

Figure 4: A multithreading version of Remove

prove performance for applications where this is the bottleneck.

A second advantage of a multithreaded architecture is that a cache miss in one of the clusters will only stall that single cluster. All other clusters are unaffected. In a processor with a single control flow, the complete processor must stall on a cache miss. This can become problematic for very wide architectures with many cache accesses per cycle.

4 Remove scheduler

Remove is a statically scheduled architecture. Its performance therefore relies heavily on the compiler. The back-end of the Remove compiler is responsible for scheduling the instructions on the Remove core and for the register allocation. In the following subsections we will analyze the problem of scheduling for Remove and we will propose an algorithm to solve this problem.

4.1 Analysis of the scheduling problem

Scheduling for Remove involves the placement of operations and variables in the data dependence graph (DDG) of the application on the two-dimensional space-time space. This space can be visualized as a graph, as can be seen in Figure 5. The size of the space side is the number of units in the target processor, and the size in the dimension time is the schedule length in cycles. Each node represents an arithmetic unit or register file at a certain time point. Note that the individual registers inside the register files are not reflected in this model.

The task of the scheduler is to map each operation in the DDG to a node in the space-time graph. Identity operations must be scheduled if a variable in the DDG is not immediately used after its creation. Just as all nodes (the operations) in the DDG must be mapped to nodes in the space-time graph, all edges in

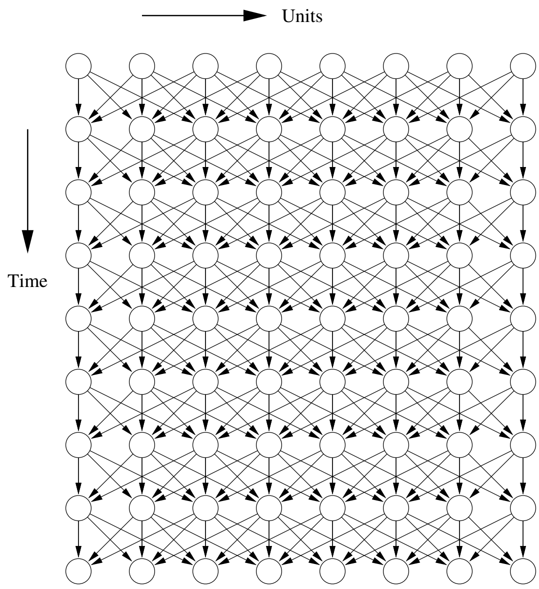

Figure 5: Graphic representation of the space-time space of a Remove architecture.

the DDG must be mapped to edges in the space-time graph. This reflects the requirement that an operation can only be scheduled on a certain unit if its operands are available at a connected unit in the previous cycle. The scheduler should try to make the shortest possible schedule. This translates to mapping the DDG on a space-time grid with a minimal size in the time direction.

Figure 6 contains an example of scheduled code for the machine model shown in Figure 5. In this example we assume that the shift and multiply operations are not pipelined and take two respectively three cycles on only one unit. The immediate values take one unit for one cycle to be loaded. The nodes with arrows are identity (move) operations, which are required if the live range of a variable does not end at a

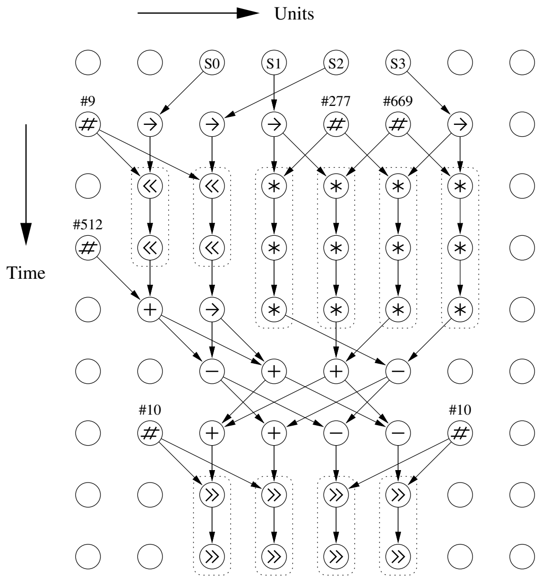

Figure 6: The DDG of an 4-point inverse DCT algorithm mapped onto the machine model of figure 5.

certain cycle. The code is taken from a 4-point inverse discrete cosine transform (DCT) algorithm. The DCT is a building block in many media applications.

Scheduling for Remove can be compared to scheduling for a clustered VLIW architecture. There are however two major differences. First, the register files in Remove are partially distributed over the arithmetic units. If the register of an arithmetic unit is occupied, its ALU is not available for operations. Practically, this means that unit allocation and register allocation for the registers in the arithmetic unit must be integrated. The only alternative is to write each result to the register file immediately to free the ALU. This is however not acceptable because of the limited number of register file ports. The second difference between Remove and a clustered VLIW is the interconnect pattern. In clustered VLIWs the clusters are discrete. Units in different clusters are not connected with the regular interconnections. The only possible communication between two clusters is a dedicated inter-cluster bus. In Remove, the units are not grouped into discrete clusters. Each unit is a cluster by itself, with limited connectivity to other clusters. The regular interconnections must also be used for communication between remote units. Remove therefore lacks a hierarchy level compared to clustered VLIWs.

These two differences make scheduling for Remove harder than for a clustered VLIW. While schedulers for clustered VLIWs are often split in a clustering phase, a scheduling phase and a register allocation phase, this is not an option for the Remove scheduler. The scheduler for Remove therefore combines these three phases in one algorithm.

4.2 Scheduling algorithm

The Remove scheduler is a cycle-based list scheduler which combines the assignment of operations to unit and time slots (nodes in the unit-time graph), and the allocation of the registers in the arithmetic units. The allocation of the registers in the register files is done in a post-scheduling phase. We will now describe a simplified version of the main scheduling algorithm in which we assume that all operations take one cycle on only one unit. It is however not hard to extend the algorithm and remove this constraint.

The algorithm converts the scheduling problem into the problem of finding the maximum weighted match in a bipartite graph. This problem is well-known as the assignment problem and can be solved by the *Hungarian algorithm* [12]. In the scheduling algorithm, the nodes in one section of the bipartite graph represent the arithmetic and register file units. The other section of the graph is reserved by the variables and operations to which units must be assigned.

The main scheduling algorithm of Remove consists of the following steps:

1. Cycle $c := 0$.
2. Create a bipartite graph G with units in one section and data/operations in the other section.
3. Add edges in G from all variables live after cycle $c - 1$ to their reachable units.
4. Assign a (heuristic) weight to all edges in G.
5. Calculate the maximum weighted perfect match of G and calculate cost C.
6. Create priority list L with data-ready operations.
7. Take highest priority operation X and remove X from L.
8. Copy G to G' and add X to the data/operations section of G'.
9. Add weighted edges in G' from X to all units on which X can be executed.
10. Remove dead variables from G'.
11. Calculate maximum weighted perfect match of G' and calculate cost C'.
12. If there exist a perfect match and C' is better than C, then assign G' to G and assign C' to C.
13. Repeat from step 7 until L is empty.
14. Assign data and operations to units in cycle c according to the best match found for G.
15. $c := c + 1$
16. Repeat from step 2 until all operations are scheduled.

In step 3–5, the algorithm starts finding a base solution for the current cycle. It does so by scheduling an identity operation (allocating a node in the bipartite graph) for all live variables. This is done to ensure that a unit (register) can be allocated for these variables further in the process. Edges are drawn to all units on which the identity operation can be scheduled, i.e. on all units that are reachable from the unit on which the variable was scheduled in the previous cycle. The goal of the assignment of weights to the edges (step 4) is to guide the bipartite matcher towards solutions that bring together variables that will be used by the same operation in the near future. Figure 7 shows the bipartite graph that may arise when five variables have to be scheduled on five units. The matching found for this graph will used as the base solution to which we compare the solutions found after adding a new operation.

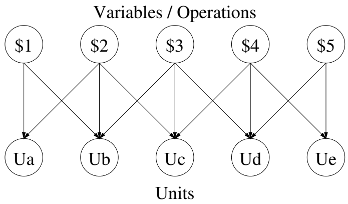

Figure 7: Initial bipartite graph for the remove scheduling algorithm. All live variables have edges to all reachable units.

In the loop 7–13 we try to add new operations to the cycle while preserving the assignments that have already been made. From the list of unscheduled operations, we take the operation with the highest priority and whose dependencies are resolved. This operation is added to the bipartite graph. Edges are drawn to all units on which the operation can be scheduled, i.e. the units that are reachable in this cycle by all operands. The edges get weights from the same heuristic that was also used in step 4. We remove all variables that will be dead after this cycle because the currently considered operation is their last usage. Figure 8 shows the resulting graph after we add an operation $O1(\$2, \$3)$ to the graph from Figure 7, assuming that it ends the life range of its second operand. The operation is accepted for the current cycle if the assignment problem solver finds a match that is better than the best solution found before the operation was added.

The quality of this algorithm heavily depends on the heuristics that are used. Good heuristics bring together variables that will be operands of the same operation in the near future and hinder the assignment of operations which block the placement of more important operations in the next cycles. Three important heuristics in the algorithm are:

- In step 6 the data-ready operations are put in a priority list. The order of this list decides which operation is tried first. A good heuristics gives precedence to operations on the critical path. Optionally, operations that end the life time of variables may be favored to reduce register pressure.
- The weights assigned to edges in the steps 4 and 9 influence the placement of the variables by the bipartite matching algorithm. The heuristic should bring together variables that are closely related in the data dependency graph
- The cost function used in steps 5 and 11 decides whether operations are accepted or rejected. A good cost function rewards adding an operation that generates results which will be used soon, while penalizing operations that hinders the placement of future operations which are more important.

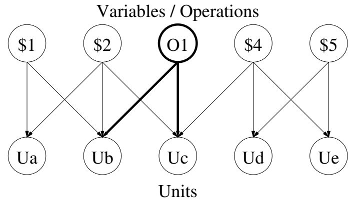

Figure 8: The scheduling graph after assigning operation $O1(\$2, \$3)$ to the current cycle, assuming that $O1$ is the last usage of $3

5 Conclusions

Remove is a new processor architecture which is designed to take full advantage of the developments in VLSI technology. It offers a huge processing capacity, while keeping the data transport network small and efficient. It is built around a processing core in which a large number of arithmetic units is ordered in a linear array. Small busses connect these units to a number of their closest neighbors. In this paper we presented the design of the architecture, and we proposed an algorithm that is capable of scheduling instructions for this architecture.

We believe that the concept of Remove is very promising and deserves further investigation. The main advantages of the Remove architecture are its regular structure that matches the requirements of modern VLSI technology, its fast communication busses, and its huge processing capacity relatively to the chip area. The regularity shortens the hardware design time and may be exploited by the scheduler. The single-source busses offer a large bandwidth, while keeping the power consumption and the transport latency at a minimum. The enormous processing power makes Remove very useful for applications with large dependency parallelism, like media applications.

References

[1] M. Bohr. Interconnect scaling - the real limiter to high performance ULSI. In *Proc. Int'l Electron Devices Meeting*, pages 241–244, 1995.

[2] Doug Matzke. Will physical scalability sabotage performance gains? *Computer*, pages 37–39, September 1997.

[3] R. P. Colwell, R. P. Nix, J. J. Donnel, D. B. Papworth, and P. K. Rodman. A VLIW architecture for a trace scheduling compiler. In *Proc. Second Int'l Conference on Architectural Support for Programming Languages and Operating Systems*, pages 180–192, October 1987.

[4] Texas Instruments Inc. *TMS320C62xx CPU and Instruction Set: Reference Guide*. Texas Instruments, Inc., July 1997.

[5] L. Gwennap. Digital 21264 sets new standard. *Microprocessor Report*, vol. 10, October 1986.

[6] K. I. Farkas, P. Chow, N. P. Jouppi, and Z. Vranesic. The multicluster architecture: Reducing cycle time through partitioning. In *Proc. 30th Ann. Int'l Symposium on Microarchitecture*, pages 149–159, December 1997.

[7] Gurindar S. Sohi, Scott E. Breach, and T.N. Vijaykumar. Multiscalar processors. In *Proc. 22nd Ann. Int'l Symposium on Computer Architecture*, pages 414–425, June 1995.

[8] E. Rotenberg, Q. Jacobsen, Y. Sazeides, and J. Smith. Trace processors. In *Proc. 30th Ann. Int'l Symposium on Microarchitecture*, pages 138–148, December 1997.

[9] Lance Hammond, Basem A. Nayfeh, and Kunle Olukotun. A single-chip multiprocessor. *Computer*, pages 79–85, September 1997.

[10] T. M. Conte, P. K. Dubey, M. D. Jennings, R. B. Lee, S. Rathnam, M. Schlansker, P. Song, and A. Wolfe. Challenges to combining general-purpose and multimedia processors into one package. *Computer*, pages 33–37, December 1997.

[11] Monica S. Lam and Robert P. Wilson. Limits of control flow on parallelism. In *ISCA-19*, pages 46–57, Australia, May 1992.

[12] Giorgio Carpaneto and Paolo Toth. Algorithm 548: Solution of the assignment problem [h]. *ACM Transactions on Mathematical Software*, 6(1):104–111, March 1980.