Collapsing floating-point operations

David Defour

Collapsing floating-point operations

David Defour
Équipe DALI, Laboratoire LP2A
Université de Perpignan
52 Avenue Paul Alduy,
66860 Perpignan Cedex, France
Phone: +34-68-66-21-35, Fax: +34-68-66-22-87, E-mail: david.defour@univ-perp.fr

Abstract

This paper addresses the issue of collapsing dependent floating-point operations. The presentation focuses on studying the dataflow graph of benchmark involving a large number of floating-point instructions. In particular, it focuses on the relevance of new floating-point operators performing two dependent operations which are similar to "fused multiply and add". Finally, this paper examines the implementation cost and critical path reduction from this strategy.

Keywords

Instruction, ILP, FMA

I. Introduction

Current high-performance microprocessors rely on hardware or software techniques to exploit the inherent instruction level parallelism of applications. As the number of transistors on a chip continues to grow, new hardware techniques need to be proposed in order to increase the performance of applications bounded by resource constrains. One technique involves exploiting data parallelism at the function unit level by executing one instruction over a large number of data. This technique is the basics of the vector processor and multimedia units in general processors. Another technique consist of developing functional units that perform multiple operations as a monolithic operation. For example the operation of the form

$$y = a + (b * c)$$

called Fused Multiply and ADD (FMA) is executed for the cost of one operation with one rounding error. Thereby reducing the overall latency of the program.

These operations that *collapsed* two basic operations, reduce the overall latency of the program by reducing the length of the dependence graph. This graph defines an execution of a computer program. The nodes of the graph characterize the operations and the arcs represent true data or control dependence between two operations. The minimum execution time of the program corresponds to the length of the longest path or *critical path* of the dependence graph.

Through the examination of various applications from benchmark, this paper studies the complex floating point operators which should be implemented in hardware. The simulation conducted in this study shows that the most relevant operators performing two floating point operations with one instruction is the FMA followed by the ADD2 operator (two consecutive additions).

The organization of this paper is as follows. Section 2 reviews related work. In Section 3, the collapsing mechanism studied in this paper is discussed. Section 4 addresses the experimental framework and assumptions. Section 5 contains the obtained results and its discussion. Section 6 will conclude the study.

II. Related work

Several studies have been conducted on strategies dedicated to reducing the length of the critical path by collapsing instructions. Collapsing for specific operations with new instruction definitions have been proposed and implemented in a number of processors. The floating-point fused multiply and add operation was used in the RS/6000 [MON 90], [OEH 90], the PowerPC [WEI 94] and recently in the Itanium and Itanium 2 [SHA 00], [MCN 03]. Similarly, a fixed point instruction Scale and Add which adds an operand to another multiplied by a factor was implemented in the Alpha ISA [ALP98].

A more general scheme able to collapse a dependent pair involving fixed point arithmetic and logical instructions was proposed by Philips and Vassiliadis in [VAS 93]. The proposed solution takes into account a general CISC instruction set that includes the functionality of RISC instruction sets. Another device that includes the majority of fixed point operations was proposed in [PHI 94]. They later investigated the potential of collapsing up to three dependent instructions in [SAZ 96].

Recently, a solution interpreting sequences of dependent instructions as functions was introduced in [YEH 04]. This solution translates fixed point instruction into a hardwired function. The Chimaera [YE 00] architecture is also related to the collapsing mechanism since it is based on reconfigurable units with shift between them. Therefore resulting in the ability of this architecture to perform instruction like addition combined with a shift.

However there have been no previous studies conducted on collapsing floating-point operations. Therefore the models used in this study are different from previous studies in that they only consider dependent floating-point instructions from the same basic block.

III. Collapsing mechanism

This section defines the collapsing mechanism. The collapsing of data dependences can be applied to any sequence of dependent instructions. For instance in the following sequence :

1. F1 = F2 * F3
2. F4 = F1 + F5

a data dependency between instruction 2 that waits for the result of instruction 1 is noticed. When the multiplication and the addition takes 5 cycles, the final result will be available after 10 cycles. However when an architecture with an operator that performs a multiplication followed by an addition (FMA) is examined, the result is delivered in 5 cycles. Therefore the process proves to take half the time to execute this sequence of instruction. This mechanism works with any kind of instruction, arithmetics, memory input/output, etc.

Floating-point operations are usually non commutative and non associative, which means that collapsing floating-point instructions might change the result. This study puts asside accuracy objective to focus on collapsing instructions. The collapsing of dependent pairs of floating-point operations of the following types are enabled: addition, subtraction, multiplication and division.

The collapsing of instruction between consecutive and non-consecutive instructions were considered as long as they remained in the same basic block. A basic block is characterized by consecutive instructions without instructions involving a split in the instruction flow, for instance a branch or a jump. When a flow of instruction in a basic block is studied, not all dependent instructions qualify for collapsing. Solely instructions that have their results used by one instructions can be collapsed. This occurs when the intermediate results is needed by more than one instruction. Hence this result is stored in a register to be used by two or more instructions and therefore cannot be collapsed.

Among the benefits of merging two dependent instructions, the reduction of the data dependency graph is the most important. In addition, this process also reduces the stress over the register file as it requires less register to store intermediate results. Another benefit from collapsing instructions is that it automatically reduces the number of operation to perform a certain task and therefore reduces the code size followed by the instruction cache misses.

However, the drawback of this approach includes the chances of one part of the operator not being used when the operator is implemented in a hardware. For example on the Itanium, when solely a multiplication is required from the FMA unit, the adder remains unused.

Name	Description	Number of fp operations			
		$+$	$-$	$*$	$/$
168.wupwise	Physics / Quantum Chromodynamics	76	42	176	15
171.swim	Shallow Water Modeling	364	151	242	15
172.mgrid	Multi-grid Solver: 3D Potential Field	511	21	104	6
173.applu	Parabolic / Elliptic Partial Differential Equations	3591	1458	4252	272
177.mesa	Graphics Library	347	102	586	27
178.galgel	Computational Fluid Dynamics	871	321	1431	100
179.art	Image Recognition / Neural Networks	253	14	247	12
183.equake	Seismic Wave Propagation Simulation	127	58	236	18
188.ammp	Computational Chemistry	479	330	930	42
189.lucas	Number Theory / Primality Testing	652	623	816	7
191.fma3d	Finite-element Crash Simulation	546	337	1042	78
301.apsi	Meteorology: Pollutant Distribution	1276	904	2376	622

TABLE I
BENCHMARK CHARACTERISTICS. A STATIC ANALYSIS A EACH BENCHMARK GIVES US THE NUMBER OF FLOATING-POINT OPERATIONS.

IV. Simulation framework

To measure the impact of collapsing on parallelism, a specific toolset was developed to extract the dataflow graph (DFG) from Alpha assembly instructions [ALP98]. The development of these tools demonstrates that adding a collapsing level within the compiler can be done without problems. The test set includes thirteen program from the benchmark from the SPEC2000fp described in table I.

The simulation process is divided into 3 stages: First the information about the program are collected with the pixstat software from the Atom toolset [EUS 95]. This tool anotates the assembly code with statistical information about the time taken by each basic block. Next, data

dependencies in the trace were analyzed and the dataflow graph (DFG) for the trace was built. Finally, information and statistics concerning floating-point operations from the DFG and the trace execution were collected.

A. Benchmark

This study used programs from the SPEC2000fp benchmarks. The benchmarks were compiled using the Compaq Alpha compiler with full optimizations (-fast). For each benchmark the number of floating-point operations and the number of instructions were assigned. The data was a result of a static analysis of assembly code.

V. Simulation Results

This section contains results of the experiments conducted. Table I reports the number of floating point addition, subtraction, multiplication and division for each program of the benchmark. Table II exhibits the number of collapsable operations as defined in section III from a static analysis of the floating point operations. The bolded number corresponds to the maximum number per benchmark which represents the highest potential presence of a collapsed operator in a benchmark. Table III represents the percentage of coverage by a collapsed operator over single operators per benchmark. In both tables the collapsed operators are presented as follows: the first operator represents the last operation performed and the second operator symbolises the first operation performed, for example the FMA is represented by $+*$.

Name	$++$	$+*$	$+/$	$*+$	$**$	$*/$	$/+$	$/*$	$//$
168.wupwise	38	**74**	0	42	27	0	7	4	0
171.swim	180	**184**	0	162	48	0	10	7	0
172.mgrid	**249**	98	0	91	11	0	2	0	0
173.applu	**1826**	1783	86	1797	782	7	69	109	0
177.mesa	116	**274**	0	164	71	0	10	10	0
178.galgel	358	**856**	26	415	443	21	57	36	10
179.art	73	98	0	**100**	34	0	8	3	0
183.equake	52	**119**	3	75	74	8	10	8	1
188.ammp	260	**418**	2	291	205	5	12	19	0
189.lucas	80	**402**	0	87	54	2	2	2	0
191.fma3d	215	**461**	12	266	144	11	24	32	0
301.apsi	514	**1015**	175	881	673	83	192	302	14
Total	3961	**5782**	304	4371	2566	137	403	532	25

TABLE II
Relative results from a static analysis.

A. Presence of single operator

The floating point addition and multiplication are almost equally present in the observed benchmark. The division represents 10 percent of the floating point operator.

B. Relevance of the FMA

From Table II, it is possible to observe that on average, the FMA column number 2 (a multiplication followed by an addition "$+*$") is the most frequent operator combining two basic operations. This signifies that among the software studied, frequently the multiplications are followed with an addition. However one can rarely observe two consecutive multiplications.

C. Others combinations

Surprisingly there exists other interesting collapsed operations. The ADD2 operator ($++$) is among one of them. First, observe that from Table II there are two programs (mgrid and applu) over thirteen that exhibit the highest number of this operator. From Table III, on average 58 percent of single floating point addition can be replaced with the ADD2 operator. This rate is not only the highest but also is superior to that of the FMA.

VI. Conclusion and perspectives

This study enforces the fact that the FMA is an interesting operator as it could be the most used among other collapsed operators. However the ADD2 operator also sparks interest as it can replace 58 percent of all the addition operations.

Name	$++$	$+*$	$+/$	$*+$	$**$	$*/$	$/+$	$/*$	$//$
168.wupwise	64.41	50.34	0	28.57	30.68	0	10.53	4.19	0
171.swim	69.9	48.61	0	42.8	39.67	0	3.77	5.45	0
172.mgrid	93.61	30.82	0	28.62	21.15	0	0.74	0	0
173.applu	72.33	38.34	3.23	38.64	36.78	0.31	2.59	4.82	0
177.mesa	51.67	52.95	0	31.69	24.23	0	4.2	3.26	0
178.galgel	60.07	65.27	4.02	31.64	61.91	2.74	8.82	4.7	20
179.art	54.68	38.13	0	38.91	27.53	0	5.73	2.32	0
183.equake	56.22	56.53	2.96	35.63	62.71	6.3	9.85	6.3	11.11
188.ammp	64.28	48.07	0.47	33.47	44.09	1.03	2.82	3.91	0
189.lucas	12.55	38.45	0	8.32	13.24	0.49	0.31	0.49	0
191.fma3d	48.7	47.9	2.5	27.64	27.64	1.96	4.99	5.71	0
301.apsi	47.16	44.56	12.49	38.67	56.65	5.54	13.7	20.15	4.5
Average	57.96	46.66	2.14	32.05	37.19	1.53	5.67	5.11	2.97

TABLE III
PERCENTAGE OF SINGLE OPERATOR COVERED BY THE COLLAPSED OPERATOR.

Therefore this study needs to be completed to observe the interaction of operators with one another. The behavior of the ADD2 in an environment with an existing FMA operator is especially intriguing. However before the possible integration of this operator into hardware, an implementation has to be found. When a hardware solution is found, the numerical implication of this operator should be studied as it is done in [OGI] with the benefit of the ADD2 over the TwoSum algorithm.

REFERENCES

[ALP98] Alpha architecture handbookoctober 1998.

[EUS 95] EUSTACE A., SRIVASTAVA A., ATOM: A Flexible Interface for Building High Performance Program Analysis Tools, *Proceedings of the Winter 1995 USENIX Technical Conference on UNIX and Advanced Computing Systems*, p. 303–314, 1995.

[MCN 03] MCNAIRY C., SOLTIS D., *Itanium 2 processor microarchitecture*, *IEEE Micro*, vol. 3, p. 44–55, april 2003.

[MON 90] MONTOYE R. K., HOKONEK E., RUNYAN S. L., *Design of the floating-point execution unit of the IBM RISC System/6000*, *IBM Journal of Research and Development*, vol. 34, n1, p. 59-70, 1990.

[OEH 90] OEHLER R. R., GROVES R. D., *Ibm risc system/6000 processor architecture*, *IBM Journal of Research and Development*, vol. 34, n1, p. 23–36, 1990.

[OGI] OGITA T., RUMP S. M., OISHI S., *Accurate Sum and Dot Product, SIAM J. Sci. Comput.*, to appear.

[PHI 94] PHILLIPS J., VASSILIADIS S., *High-Performance 3-1 Interlock Collapsing ALU's*, *IEEE Transactions on computer*, vol. 43, n3, p. 257–268, 1994.

[SAZ 96] SAZEIDES Y., VASSILIADIS S., SMITH J. E., The performance potential of data dependence speculation & colapsing, *Proceedings of the 29th annual IEEE/ACM international symposium on Microarchitecture*, p. 238–247, 1996.

[SHA 00] SHARANGPANI H., ARORA A., *Itanium processor microarchitecture*, *IEEE Micro*, vol. 20, n5, p. 14–43, september 2000.

[VAS 93] VASSILIADIS S., PHILLIPS J., BLANNER B., *Interlock Collapsing ALU's*, *IEEE Transactions on computer*, vol. 42, n7, p. 825–839, 1993.

[WEI 94] WEISS S., SMITH J. E., *Inside IBM Power and PowerPC*, Morgan Kaufmann Publishers Inc., San Mateo, 1994.

[YE 00] YE Z. A., MOSHOVOS A., HAUCK S., BANERJEE P., CHIMAERA: A high performance architecture with a tightly coupled reconfigurable functional unit, *Proceedings of the 27th annual IEEE/ACM international symposium on computer architecture*, p. 225–235, june 2000.

[YEH 04] YEHIA S., TEMAM O., From Sequences of Dependent Instructions to Functions: An Approach for Improving Performance without ILP or Speculation, *31th Annual International Symposium on Computer Architecture*, 2004.