A HW/SW Co-designed Programmable Functional Unit

Abhishek Deb*, Josep Maria Codina† and Antonio González*†
*Universitat Politecnica de Catalunya
†Intel Research Labs Barcelona
C. Jordi Girona 1-3, Barcelona, Spain
abhishek@ac.upc.edu, {josep.m.codina, antonio}@intel.com

Abstract—In this paper, we propose a novel programmable functional unit (PFU) to accelerate general purpose application execution on a modern out-of-order x86 processor. Code is transformed and instructions are generated that run on the PFU using a co-designed virtual machine (Cd-VM). Results presented in this paper show that this HW/SW co-designed approach produces average speedups in performance of 29% in SPECFP and 19% in SPECINT, and up-to 55%, over modern out-of-order processor.

1 Introduction

Introducing SIMD-like hardware accelerators needs to be supported by extending the ISA [1]. Applications need to be recompiled to the new ISA in order to use these hardware accelerators. A software binary translation layer tightly integrated to the underlying processor can be used to dynamically and transparently recompile the application binary. Such a HW/SW co-designed approach is often referred to in literature as a co-designed virtual machine (Cd-VM), and Transmeta Crusoe [2] is a commercial example.

In this paper, we propose a novel programmable functional unit (PFU) to accelerate general purpose application execution, in a complexity-effective way. We leverage the fact that reducing the critical path and exploiting the ILP leads to performance improvement.

Results presented in this paper show that the use of a PFU provides a significant average speedup of 29% in SPECFP and 19% in SPECINT, and speedup of up-to 55% for some benchmarks, over current out-of-order processor.

The key contributions of this paper are as follows:

- We propose a novel Programmable Functional Unit, along with a novel split-MOP execution model.
- We describe an effective algorithm to dynamically fuse instructions using a Cd-VM.

The rest of the paper is organized as follows. In Section 2, the proposed PFU is described along with its execution model and microarchitecture. Dynamic compilation techniques are discussed in Section 3. A detailed evaluation and analysis of the PFU, its design points and comparison with alternate schemes is presented in Section 4. Finally, related work is reviewed in Section 5 and we conclude in Section 6.

2 Proposed Microarchitecture

2.1 Split-Mop Model

Our split-MOP model consists of following micro-ops - a set of loads to provide inputs from memory (ld-set), a set of register moves to provide inputs from register file (mv-set), a computation macro-op (CMOP), and a store set (st-set). Figure 1 shows an example of split-MOP. Note, that irf0, irf1, irf2 in Figure 1b indicates the IRF (Internal Register File, discussed later) registers. We have also considered a version with no mv-set.

```
ld rcx = [rax, 8]
add rdx = rcx, rax
add rsp = rbx, 1
sub rbp = rsp, rdx
st [rbp, 4], rax
```

(a) Micro-ops before fusion

```
ld irf0, rcx = [rax,8] : ld-set
mov irf1, irf2 = rax, rbx : mv-set
cmop rdx, rsp, rbp
st [rbp, 4], rax

irf* : Internal Register File Tag
```

(b) Macro-op after fusion

Fig. 1: Split-MOP Model

2.1.1 Computation Macro-Op (CMOP)

The ld-set and mv-set is followed by a CMOP, which contains information of the fused micro-ops. CMOP consists of a unique identifier, and destination registers. Transient registers are not reflected in CMOP's destination register. CMOP does not contain any source operands, because, it reads input values from the IRF, but it writes directly to the physical register file. Hence, the number of destination registers in CMOP is constrained by the number of write ports in the physical register file.

2.1.2 Execution Model

The CMOP issues when all the loads in the ld-set and moves in the mv-set have issued. CMOP is executed in the PFU and a typical execution pipeline of the split-MOP of Figure 1 is shown in the Figure 2.

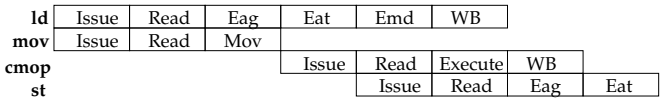

Fig. 2: Execution Pipeline

The execution pipeline of the split-MOP as described in Figure 1 is illustrated in Figure 2. The four execution pipelines correspond to load, mov, CMOP and store respectively of

Manuscript submitted: 09-Jun-2011. Manuscript accepted: 26-Sep-2011. Final manuscript received: 27-Sep-2011

Figure 1b. The pipeline stages Eag, Eat and Emd stands for address generation, address translation and memory disambiguation respectively. Moreover, CMOP is executed back-to-back with its ld-set and mv-set.

CMOP dependence with the ld-set and mv-set is built at runtime. This is achieved using a hierarchical issue queue model as described in Section 2.2.5.

2.2 PFU Microarchitecture

2.2.1 Programmable Functional Unit

We propose a PFU which has two major components: 1) Distributed Internal Register File (IRF), and (2) a grid of Processing Elements (PE). Data flows from one row to the following in the grid of PEs as shown in Figure 3. Note, that there are no latches between the PEs of two different rows. The inputs required by each micro-op in the grid of PEs is provided by the IRF.

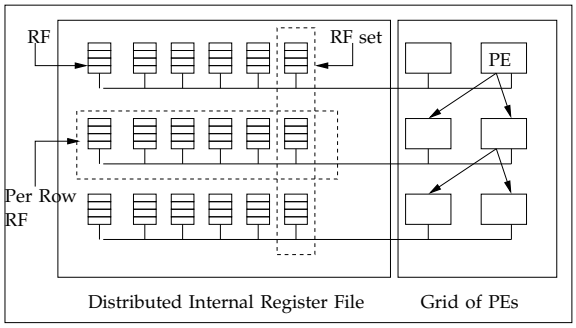

Fig. 3: Programmable Functional Unit overview

2.2.2 Distributed Internal Register File

The proposed PFU with six PEs (2 columns, 3 rows) requires up to twelve read ports to execute all the micro-ops of CMOP simultaneously. Providing so many read ports to the physical register file is certainly not complexity-effective. Hence, in order to deal with this, we propose a separate register file (IRF) contained in the PFU.

Moreover, inputs for multiple CMOPs are brought into the IRF from the L1 D-cache and from the physical register file. For successful execution, the IRF should have sufficient capacity, in order to hold the inputs for multiple CMOPs. A lack of this capacity can significantly handicap out-of-order execution, by causing unnecessary stalls. A centralized internal register file with sufficient capacity with 12 read ports is also not affordable.

This internal register file is distributed in order to provide sufficient bandwidth, as shown in Figure 3. IRF contains multiple register file sets, each of which is allocated to a MOP in the dispatch stage. A register file set contains replicated copies of register file, one copy corresponding to each row. Each register file has 4 entries and has 4 read ports and 4 write ports. Recollect, that the CMOP writes to the physical register file. Hence, the write ports on IRF are used by the ld-set and mv-set only. More details on PFU can be obtained from [3].

There are 5 different register file sets, so inputs for 5 different MOPs can be stored at the same time. Hence, the total size of this IRF is 60 (number of entries per RF*number of rows*number of RF sets = 4*3*5). Dispatch stalls in case a register file set cannot be allocated to the MOP. It is obvious from such a distributed organization that a PE could access only the register file of the row that it belongs to and to that of the MOP that is currently being executed.

Note, that total number of entries both in the ld-set and the mv-set are constrained by the size of one register file. The values in the IRF are discarded only when CMOP is successfully executed.

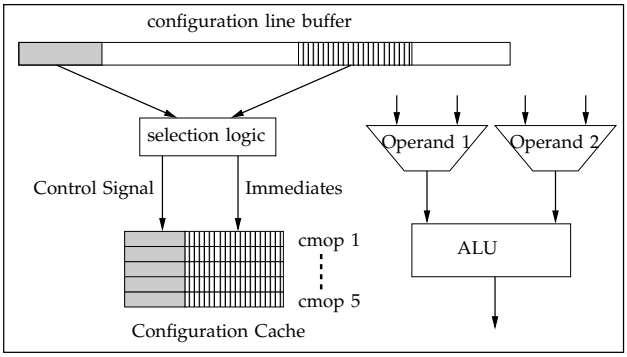

Fig. 4: Processing Element

2.2.3 PE and Configuration Cache

Figure 4 provides a deeper look into the PE. Each PE contains 1) an ALU, which is connected to the ALUs of following rows, and 2) a configuration cache. Configuration cache holds configuration of 5 CMOPs in a distributed manner. The configuration contains pre-decoded control signals of all the fused micro-ops pertaining to the CMOP as shown in the Figure 5.

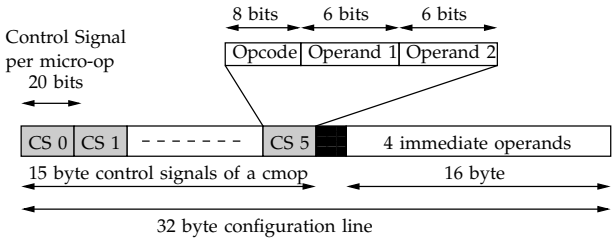

Fig. 5: Configuration Line

A direct access is made to the L1 I-cache to read in the configuration line corresponding to a CMOP. The configuration line is then distributed to all the PEs. Each PE contains also a line buffer to store the configuration line. The PE then selects the appropriate micro-op control signal and immediate operand, if any, from the configuration line buffer. The control signals are stored in the configuration cache.

The configuration line in the line buffer is discarded in the following cycle. We propose a single cycle selection logic. Hence, 3 cycles are sufficient to ensure that the configuration corresponding to a CMOP is properly distributed. A request for loading the configuration is, hence, sent in the rename stage. CMOP issue stalls if the configuration is not yet loaded. Due to the spatial locality of instructions we don't discard the configuration. The PFU can hold up to 5 different configurations at any time. The configurations are managed using a simple LRU scheme.

2.2.4 Bypass Network

To support back-to-back execution, all the 6 PEs should receive source operands from the bypass. The PEs, however, receive

inputs only from the 2 load units (LDUs) and 2 ALUs. In the evaluation Section 4, we however show that not all the 6 PEs need the source operands to be bypassed. For a 2x3 grid a bypass network to 4 PEs is more than sufficient.

On the other hand, a significant fraction of execution cycle of an ALU in a modern out-of-order processor is consumed by the destination operand forwarding [4], [5]. Hence, in order to support a PFU that collapses three ALUs and execute with low latency, we remove the forwarding logic from PFU to other ALUs, and dedicate this fraction of execution cycle completely to execution. Our studies indicate that such a constraint has negligible impact on performance.

2.2.5 Control Issue Queue

Loads of the ld-set go to the traditional Issue queue, and an entry in the control issue queue is allocated for each ld-set. The control issue queue entry contains issue queue tags of all the loads in the ld-set. The same holds true for all the moves in the mv-set. This hierarchical issue queue model is described below, and illustrated in Figure 6.

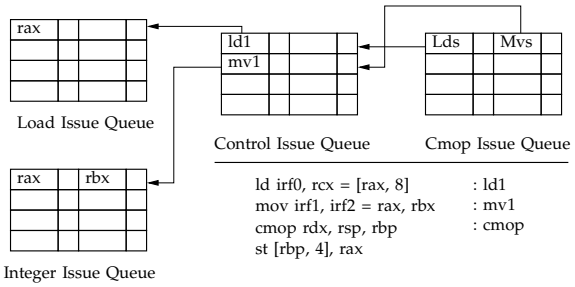

Fig. 6: Hierarchical Issue Queue Model

Ld1 is the tag associated with the load in the load set. Control issue queue entry corresponding to tag Lds depends upon ld1 issue queue entry, as shown by a backward arrow in Figure 6. Similarly, CMOP depends upon the Lds and Mvs control issue queue entry tags. Ld1 issue queue tag is broadcast to the control issue queue and Lds issue queue tag is broadcasts to CMOP issue queue, where CMOPs are held. CMOP's dependence with ld-set and mv-set entry is built at runtime using information encoded in the CMOPs. Such a model ensures that CMOP issues only when both the ld-set and the mv-set have issued, without having the need of explicit source operand encoding in a CMOP.

3 Code Generation

The code generation step is shown in Figure 7. In the fusion step, the micro-ops are considered once for fusion. For further details of code generation process, please refer [3].

4 Performance Evaluation

4.1 Experiment Methodology

We have implemented the proposed PFU model including the Cd-VM in a modified version of PTLSim[6]. The simulated processor is a modern 4-way out-of-order processor. The baseline and proposed microarchitecture details can be obtained from [3]. Moreover, [3] also contains detailed analysis; in this section we present a condensed version. We could not simulate the functional behaviour of gcc correctly, hence its performance is not included in the results.

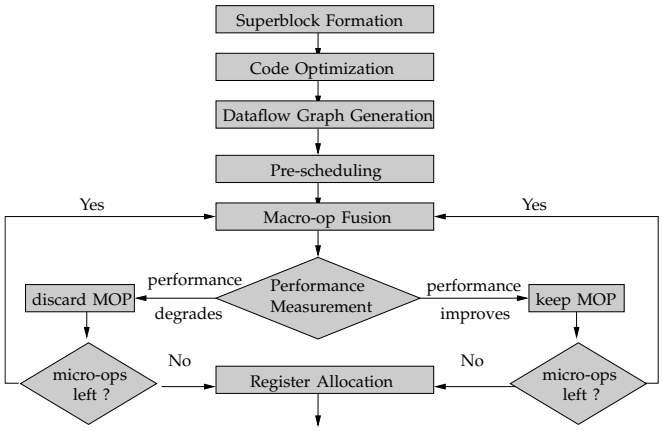

Fig. 7: Code Generation Flow Chart

4.2 Overhead of Co-designed Virtual Machine

To measure the overhead of Super Block Translation we used estimates provided in [7], [8]. DAISY [8] reported an estimate of nearly 4000 source instructions to optimize a single source instruction. They however, mention that this a very conservative estimate and quote a reasonable estimate to be 1000 source instructions to optimize a single source instruction. S. Hu [7] measured the overhead to be 1000 x86 instructions per x86 instruction, hence we use the same to measure overhead.

Table 1 shows the overhead of Super Block Translation, corresponding to 100 million dynamic instructions, for the benchmarks in terms of x86 instructions. Note that, the overhead is due to hot code translation and optimization, which have a small static footprint. Cold code, however, are converted using dual-mode decoders proposed in [7]. Hence, the startup overhead of Super Block translation is smaller than one percent.

4.3 Impact of Microarchitectural Constraints

PFU as described earlier is a grid of PEs. We try to vary the number of columns in the grid from 1 to 4, but keeping the number of rows fixed to 3 as shown in Figure 8. In SPECINT the 2x3 grid is the best performing configuration. For SPECFP, the 2x3 grid provides performance close to that provided by a 4x3 grid. Therefore, we choose 2x3 grid configuration for PFU.

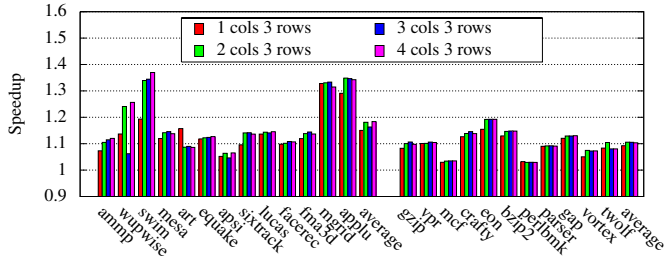

Fig. 8: Impact of grid size

Note, however, that in some cases increasing the number of columns in the grid (e.g. wupwise, applu, twolf) results in a lower performance improvement. The main reason for this is the fact that the heuristic for fusion, fuses as many micro-ops possible. Thus, in some cases, micro-Ops that are independent are also fused,and so delay in input for one micro-op delays the execution of the CMOP.

SPECFP	ammp	wup	swim	mesa	art	equake	apsi	sixtr	lucas	facer	fma3d	mgrid	applu.
	1818 K	758 K	209 K	2086 K	366 K	1144 K	2347 K	412 K	84 K	644 K	3955 K	81 K	5859 K
SPECINT	gzip	vpr	mcf	crafty	eon	bzip2	perlbmk	parser	gap	vortex	twolf		
	460 K	1292 K	281 K	4756 K	2463 K	126 K	1605 K	2797 K	2349 K	3223 K	1517 K		

TABLE 1: Overhead in terms of x86 instructions

4.4 Impact of Mov-set

From Figure 8 we have concluded that a PFU configuration with 2 columns and 3 rows is a good trade-off. Moreover, in [3] we conclude that only four of these six (2*3) PEs need input from bypass or the Physical Register File. This provides us with an opportunity of removing mov-set entirely. The number of read ports in Physical Register File is eight and can satisfy the need of PFU. However, Load-set are still required and they write their outputs to both the Physical Register File and the Internal Register File. Consequently, the cmop contains both multiple source and destination operands, subject to a total of four operands.

The benefit of removing the mov-set is significant and is shown in Figure 9. On an average, we obtain a performance benefit of 29% and 19% for SPECFP and SPECINT, respectively. This is a speedup of nearly 10% over the version with mov-set. This benefit is primarily due to gain in back-to-back execution between the producers of cmop and the cmop. Furthermore, by removing mov-set we reduce the dynamic instruction count and reduce the pressure on execution resources of the processor.

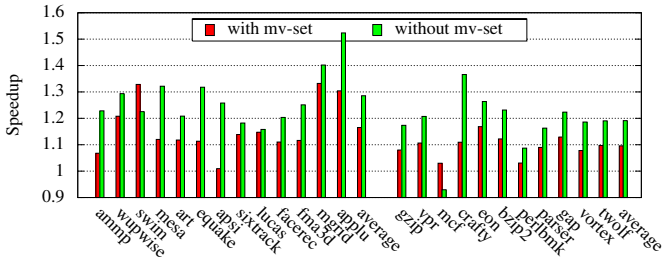

Fig. 9: Impact of removing mov-set

4.5 Comparison with alternate designs

We compared our proposed PFU with two alternate designs. In the first alternative we double the issue width and the writeback width of the processor and increase the ALUs from two to eight, in order to match with six PEs in the PFU. In the second alternative we double the issue width and the writeback width and all the functional units. Clearly, our PFU outperforms all the three alternatives as shown in Figure 10.

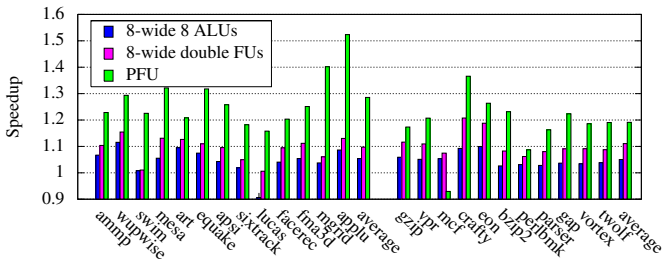

Fig. 10: Comparison with SIMD and 8-way out-of-order

5 Related Work

Previouly propose accelerators [9], [10] are designed requiring changes to the ISA and they are programmed using a static code generation scheme. Dynamic and transparent management of accelerators have been studied in the past. Most of the works [9], [11] have focused on managing accelerators with hardware-based dynamic optimizers. A couple of limited amount of work has considered the use of co-designed VM for assigning instructions to an accelerator [12], [13].

6 Conclusions

In this paper we proposed a split-MOP model that allows inputs to be provided both from memory and conventional register file. A novel PFU is proposed that executes the CMOP. Such a design exploits both ILP and instructions collapsing to gain performance. We obtain average speedups of 29% in SPECFP and 19% in SPECINT, and up-to 55%.

References

[1] "SSE extension : Intel IA 64 and IA-32 Architectures Software Developer's Manual," 1997.
[2] A. Klaiber, "The technology behind Crusoe Processors," 2000.
[3] A. Deb, J. Codina, and A. González, "A Co-designed HW/SW Approach to General Purpose Program Acceleration using a Programmable Functional Unit," in *IEEE 15th Workshop on Interaction between Compilers and Computer Architecture*, 2011.
[4] E. Fetzer, M. Gibson, A. Klein, N. Calick, C. Zhu, E. Busta, and B. Mohammad, "A fully bypassed six-integer datapath and register file on the itanium-2 microprocessor," in *IEEE Intl. Journal of Solid-State Circuits*, 2002.
[5] S. Palacharla, N. Jouppi, and J. Smith, "Complexity-effective superscalar processors," in *IEEE Intl. Symp. on Computer Architecture*, 1997.
[6] M. Yourst, "PTLsim: A Cycle Accurate Full System x86-64 Microarchitectural Simulator," in *IEEE Intl. Symp. on Performance Analysis of Systems and Software*, 2007.
[7] S. Hu and J. E. Smith, "Reducing startup time in co-designed virtual machines," in *In Proc. of the 33rd Annual International Symposium on Computer Architecture*, 2006.
[8] K. Ebcioglu and E. Altman, "DAISY: Dynamic compilation for 100% architectural compatibility," in *IEEE Intl. Symp. on Computer Architecture*, 1997.
[9] N. Clark, M. Kudlur, H. Park, S. Mahlke, and K. Flautner, "Application-Specific Processing on a General-Purpose Core via Transparent Instruction set customization," in *IEEE Intl. Symp. on Microarchitecture*, 2004.
[10] Z. Ye, A. Moshovos, S. Hauck, and P. Banerjee, "CHIMAERA: A high-performance architecture with a tightly-coupled reconfigurable functional unit," in *IEEE Intl. Symp. on Computer Architecture*, 2000.
[11] S. Yehia and O. Temam, "From sequences of dependent instructions to functions: An approach for improving performance without ILP or speculation," in *IEEE Intl. Symp. on Computer Architecture*, 2004.
[12] S. Hu, I. Kim, M. Lipasti, and J. Smith, "An approach for implementing efficient superscalar CISC processors," in *IEEE Intl. Symp. on High-Performance Computer Architecture*, 2006.
[13] A. Deb, J. Codina, and A. Gonzalez, "SoftHV : A HW/SW Co-designed Processor with Horizontal and Vertical Fusion," in *Proceedings of the 8th ACM International Conference on Computing Frontiers*, 2011.