PowerPC Compiler Backend for the Molen Programming Paradigm

Elena Moscu Panainte, Koen Bertels, and Stamatis Vassiliadis
Computer Engineering Laboratory
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4, 2600 GA Delft, The Netherlands
Phone: +31 15 2786249 Fax: +31 15 2784898
E-mail: {elena|koen|stamatis}@ce.et.tudelft.nl

Abstract— **The Molen Programming Paradigm is a sequential consistency paradigm for programming CCMs (Custom Computing Machine) possibly including a general purpose computational engine(s). The paradigm allows for parallel and concurrent hardware execution and it is intended (currently) for single program execution. It requires only a one time architectural extension of few instructions to provide a large user reconfigurable operation space. The Molen machine organization has been implemented on Virtex II Pro Platform, with an IBM PowerPC 405 processor immersed into the FPGA fabric. For the validation of the proposed programming paradigm, we develop a PowerPC compiler backend, integrated into Delft WorkBench compiler. In order to illustrate the complexity of generating code for a modern processor such as PowerPC 405, several issues regarding special features, user-programming model, the impact of the operating systems and the PowerPC EABI are presented.**

Keywords—**retargetable compiler; Custom Computing Machine (CCM); PowerPC; compiler backend**

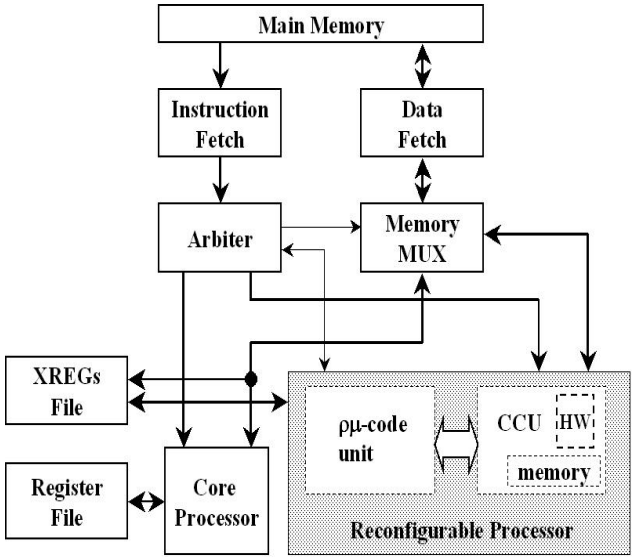

Fig. 1. The Molen machine organization

I. Introduction and Background

The development of architectural improvements or innovations is a complex process as it deals with a large number of highly interconnected factors. An improvement in one component does not necessarily result in an improved system performance. This complexity increases considerably as heterogeneous architectures (e.g. ASIC, FPGA's) are included. Such an approach is becoming increasingly popular (e.g. [1], [2], [3], [4]) as it allows developers to better partition and manage their projects (e.g. [5], [6], [7], [8] and [9]). Exploiting the full potential of these future architectures is not a trivial task. In [10] a programming paradigm called the Molen Programming Paradigm is presented that facilitates the development of such systems.

In the remainder of this section we explain the basic approach. We then introduce a specific implementation of the Molen architecture using the Xilinx Virtex-II Pro with a PowerPC 405 processor immersed into the FPGA fabric. We then present the compiler extensions required for the Molen Programming Paradigm and discuss the specific characteristics of the target PowerPC 405 Processor.

The organization of the target CCM is the Molen machine organization[4] presented in Figure 1. The main components are the Core Processor - which is a GPP (General Purpose Processor), and the Reconfigurable Processor - implemented in the FPGA. The arbiter performs a partial decoding of the instructions fetched from the main memory and issues them to the corresponding execution unit. The parameters for the FPGA reside in the Exchange Registers.

Programing the target CCM is performed under the Molen Programming Paradigm[10] which is a sequential consistency paradigm for CCMs possibly including a general purpose computational engine(s). The paradigm allows for parallel and concurrent hardware execution and it is intended (currently) for single program execution. It requires only a one-time architectural extension of few in-

structions to provide a large user reconfigurable operation space. In this article the relevant extensions are:

- Two instructions for controlling the reconfigurable hardware, namely:
  - SET $< address >$: actually perform the hardware configuration as stored in memory from the referred address location. The information about the configuration microcode length is embedded inside the microcode itself.
  - EXECUTE $< address >$: for controlling the executions of the operations on the reconfigurable hardware. The address sequence referred by this instruction contains the microcode to be executed on the CCU configured in the SET phase.
- Two move instructions for passing of values to and from the GPP register file and the reconfigurable hardware. More specially:
  - MOVTX $XR_a \leftarrow R_b$: (move to XR) used to move the content of general purpose register R_b to XR_a.
  - MOVFX $R_a \leftarrow XR_b$: (move from XR) used to move the content of exchange register XR_b to GPP register R_a.

The paradigm also requires for the reconfigurable hardware to have associated a special set of registers - Exchange Registers(XRs) for passing values to/from GPR (General purpose Register). Finally, it is noted that every user is provided with an arbitrary number of functions that can be performed on the reconfigurable hardware.

II. Virtex-II Pro and PowerPC 405 Processor

The Virtex-II Pro family contains platform FPGAs for designs that are based on IP cores and customized modules. The family incorporates up to four IBM PowerPC RISC 405 processor blocks, with the following main features:

- embedded 300+ MHz Harvard Architecture Block
- low power consumption: 0.9 mW/MHz
- five-stage data path pipeline
- hardware multiply/divide unit
- thirty-two 32-bit General Purpose Registers
- 16 KB two-way set-associative instruction cache
- 16 KB two-way set-associative data cache
- memory management unit (MMU)
  - Variable page sizes (1 KB to 16 MB)
- dedicated on-chip memory (OCM) interface
- supports IBM CoreConnect" bus architecture
- debug and trace support
- timer facilities

Virtex-II Pro devices incorporate large amounts of 18Kb Block SelectRAM+ memory. The available memory resources for Virtex-II Pro XC2VP20 is around 300 Kb while for XC2VP50 is around to 700 Kb. OCM controllers provide dedicated interfaces between Block SelectRAM+ memory and processor block instruction and data paths for high-speed access routing resources.

These features make the The Virtex-II Pro platform suitable for the implementation of the Molen Machine Organization. A key element is the implementation of the arbiter which is described in detail in [11].

III. Compiler backend Development for PowerPC 405 Processor

In the next subsections, we briefly describe the key aspects for developing a compiler backend in order to generate code for PowerPC 405 processor included in the Virtex-II Pro. We first describe the implemented compiler extensions for the Molen Programming Paradigm and continue with the required extensions for developing a pure PowerPC backend compiler.

A. Compiler Extension for Molen Programming Paradigm

The compiler [12] currently relies on the Stanford SUIF2 (Stanford University Intermediate Format)[13] for the front-end and the Harvard Machine SUIF[14] backend framework. The last component has been designed with retargetability in mind. It provides a set of backends for GPPs, powerful optimizations, transformations and analysis passes. These are essential features for a compiler targeting a CCM. We have currently implemented the following extensions for the Molen Programming Paradigm:

- Code identification: for the identification of the code mapped on the reconfigurable hardware, we added a special pass in the SUIF front-end. This identification is based on code annotation with special pragma directives (similar to [6]). In this pass, all the calls of the recognized functions are marked for further modification.
- Instruction Set extension: the Instruction Set has been extended with SET/ EXECUTE and MOVTX/MOVFX instructions at both MIR (Medium Intermediate Representation) level and LIR (Low Intermediate Representation) level.
- Register file extension: the Register File Set has been extended with the XRs. The register allocation algorithm allocates the XRs in a distinct pass applied before the GPR allocation; it is introduced in Machine SUIF, at LIR level.
- Code generation: code generation for the reconfigurable hardware is performed when translating SUIF to Machine SUIF IR, and affects the function calls marked in the front-end.

An example of the code generated by the extended compiler for the Molen Programming Paradigm is presented in Figure 2. In the first part, the C program is given. The function implemented in reconfigurable hardware is annotated with a pragma directive named *call_fpga*. It has incorporated the operation name, *op1* as specified in the

Register	Type	Usage
R0	Volatile	Language specific
R1	Dedicated	Stack Pointer (SP)
R2	Dedicated	Read-only small data area anchor
R3 - R4	Volatile	Parameter Passing/ return values
R5 - R10	Volatile	Parameter Passing
R11 - R12	Volatile	
R13	Dedicated	Read-write small data area anchor
R14 - R31	Nonvolatile	
Fields CR2 - CR4	Nonvolatile	Condition Register
Other CR fields	Volatile	Condition Register
Other registers	Volatile	

TABLE I
POWERPC EABI REGISTER USAGE

FPGA description file. In the central part of the picture,the code generated by the original compiler for the C program is depicted. The pragma annotation is ignored and a normal function call is included. The last part of the picture presents the code generated by the compiler extended for the Molen Programming Paradigm; the function call is replaced with the appropriate instructions for sending parameters to the reconfigurable hardware in XRs, hardware reconfiguration, preparing the fix XR for the microcode of the EXECUTE instruction, execution of the operation and the transfer of the result back to the GPP. The presented code is at MIR level and the register allocation pass has not been applied.

B. PowerPC Compiler backend

In order for one application to utilize external and/or underlying software or hardware, a binary interface - called Application Binary Interface (ABI) has to be defined. For example, many applications have to include a set of libraries (e.g. math) that are compiled using a number of platform dependent conventions. One such set of conventions proposed for PowerPC 405 is the Embedded Application Binary Interface (EABI) with the goal of reducing memory usage and optimizing execution speed, as these are prime requirements of embedded system software. The EABI describes conventions for register usage, parameter passing, stack organization, small data areas, object file, and executable file formats. A description of the key issues for the PowerPC compiler backend is presented in the rest of this section.

B.1 Register Usage

In user mode, The PowerPC 405 processor provides the following registers:

- General Purpose Registers (GPRs): 32 registers, each 32 bits wide, numbered r0 through r31;
- Condition Register (CR): a 32-bit register that reflects the result of certain instructions and provides a mechanism for testing and conditional branching; for example a branch based on the condition r3 $<$ 64 can be implemented as follows:

```
; CR has 8 fields of 4 bits each
cmplwi 3, r3, 64   ; CR3 field contain
       ; the result of the comparison
blt 3, LABEL_1  ; branch based on CR3
......
```

- Fixed-Point Exception Register (XER): a 32-bit register that reflects the result of arithmetic operations that have resulted in an overflow or carry;
- Link Register (LR): a 32-bit register that is used by branch instructions, generally for the purpose of subroutine linkage;
- Count Register (CTR): a 32-bit register that can be used by branch instructions to hold a loop count or the branch-target address;
- User-SPR General-Purpose Register (USPRG0): a 32-bit register that can be used by application software for any purpose;
- SPR General-Purpose Registers (SPRG4- SPRG7): 32-bit registers that can be used by system software for any purpose and available with read-only access
- Time-Base Registers: a 64-bit incrementing counter implemented as two 32-bit registers(TBU and TBL) with read-only access

The PowerPC EABI register usage conventions are depicted in Table I. Nonvolatile registers must have their original values preserved, therefore, functions modifying nonvolatile registers must restore the original values before returning to the calling function.

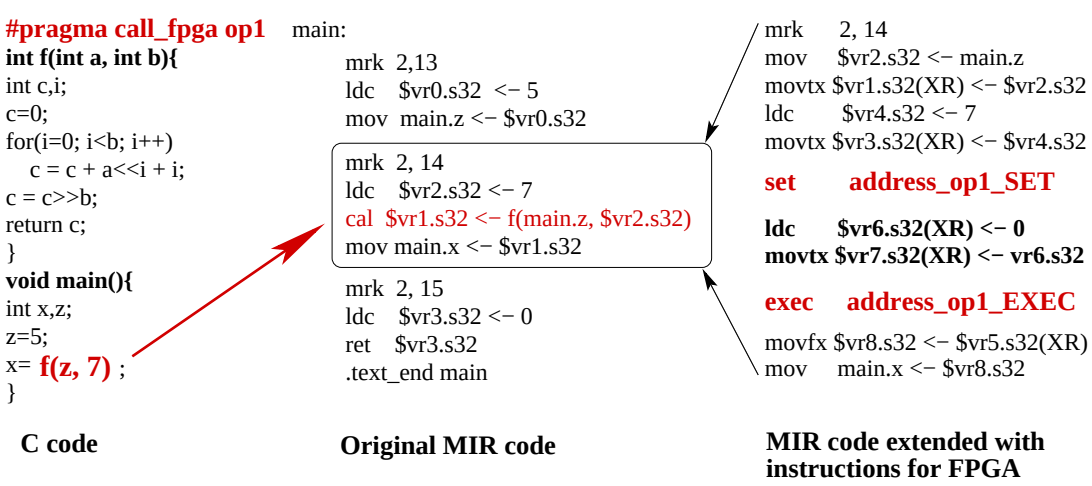

Fig. 2. Code Generation at MIR level

```
FuncX:

        mflr %r0              ; Get Link register
        stwu %r1,-88(%r1)     ; Save Back chain and move SP
        stw %r0,+92(%r1)      ; Save Link register
        stmw %r28,+72(%r1)    ; Save 4 non-volatiles r28-r31

        ................................................

        lwz %r0,+92(%r1)      ; Get saved Link register
        mtlr %r0              ; Restore Link register
        lmw %r28,+72(%r1)     ; Restore non-volatiles
        addi %r1,%r1,88       ; Remove frame from stack
        blr                   ; Return to calling function
```

Fig. 3. Function's Prologue and Epilogue

The EABI also has a construct known as the Small Data Area (SDA) designed to take advantage of the PowerPC base plus displacement addressing mode. When a 16-bit displacement fits, along with the instruction opcode, into a single instruction word than only one instruction word is required instead of the two needed to access it as a 32-bit address. This is a more memory efficient method of accessing a variable from SDA than referencing it by using a full 32-bit address. There are two such SDAs, one for read-write variables and a second for read-only variables. The small data areas are referenced by a base register loaded when the C runtime environment is initialized. R2 is the base for the read-only (const type) small data area, and R13 is the base for the read-write (nonconst type) small data area.

B.2 The Stack Frame

In addition to the registers, each function may have a stack frame on the runtime stack. The PowerPC architecture does not have a push/pop instruction for implementing a stack. The EABI conventions of stack frame creation and usage for parameter passing, nonvolatile register preservation, local variables, and code debugging are presented in Fig. 4. The following requirements apply to the stack frame:

- The address of the previous frame is stored in Back Chain Word, thereby forming a linked-list of stack frames and it is always located at the lowest address of the stack frame.
- The return address to the calling function is stored in the LR Save Word.
- In order to maintain 8-byte alignment of the stack frame, a Padding Area may be introduced to guarantee such alignment.
- In the Function Parameters Area, additional function arguments are stored when they do not fit into the designated registers R3-R10.
- When local variables are more than can be contained in the available volatile registers, they are stored in Local Variables Area.

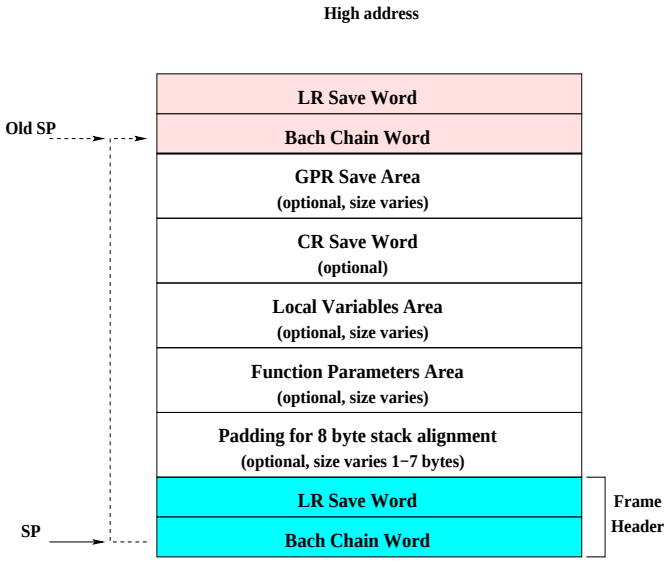

Fig. 4. PowerPC EABI Stack Frame Organization

- When the nonvolatile CR fields are modified, its contain need to be saved in the CR Save Word.
- GPR Save Area may be introduced to save nonvolatile GPR. When saving any GPR, all the GPRs from the lowest through R31, inclusive, must be saved.

The stack frame is created by a function's prologue code and destroyed in its epilogue code. In Fig. 3 is presented an example of function's prologue and epilogue.

B.3 Floating-Point Emulation

The PowerPC 405 is an integer processor and does not support the execution of floating-point instructions in hardware. System software can provide floating-point emulation support by supplying a call interface to subroutines within a floating-point run-time library. The individual subroutines emulate the operation of floating-point instructions as presented in [15]. This method requires the recompilation of floating-point software in order to add the call interface and link in the library routines.

The compiler has to manage floating point arithmetic, comparisons, loads, and stores by generating software floating point emulation (sfpe) code, rather than using PowerPC floating point instructions. In sfpe code:

- Floating point single precision scalars shall be treated as long int scalars.
- Floating point double precision scalars shall be treated as long long scalars.
- Whenever a function has a variable argument list, it shall not set condition register bit 6 to 1 (as usual for PowerPC architecture), since no arguments are passed in the floating-point registers (as there are no FPR included in PowerPC 405).

B.4 Code Selection

Code selection is typically the first backend phase and maps machine independent IR statements and operations into machine specific processor instructions. This phase is performed in Machine SUIF where each IR statement is translated into equivalent assembly instructions. For example, for the MIR instruction *mul r1, r2* $\rightarrow$ *r3*, the generated LIR set of instructions depends on the operands type as presented in Table II. For example, when both operands are of type unsigned short (2 bytes), then a mask for each operand (to take the lower 16 bits) is required and the result of the single multiplication can be placed in a 32-bit register. For integer operands, two multiplications are required when a 8 byte result is expected.

For RISC targets with homogeneous register files, the translation of each MIR instruction separately provides satisfactory results, since there are hardly complex instructions and late improvement of the selected code is still possible by means of peephole optimization.

IV. Conclusions and Future Work

In this paper we have presented the implemented compiler extensions for the Molen Programming Paradigm and determined the key issues for developing a compiler backend for the PowerPC 405 processor included in Virtex-II Pro Platform.

As future work, we intend to develop compiler optimizations in order to hide the SET instruction latency and moreover to exploit the parallelism of operations executed on the reconfigurable hardware. As the compiler can generate additional code to perform a deep profiling for a set of representative input data, we can also analyze the hardware constraints for reconfigurable hardware configurations and executions, GPP-FPGA communication and FPGA-memory communication for the Virtex-II Pro Platform.

References

[1] F. Campi, R. Canegallo, and R. Guerrieri. IP-Reusable 32-Bit VLIW Risc Core. In *Proc. of the 27th European Solid-State Circuits Conference*, pages 456–459, Villah, Austria, Sep 2001.

[2] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins. ADRES: An Architecture with Tightly Coupled VLIW Processor and Coarse-Grained Reconfigurable Matrix. In *FPL*, volume 2778, pages 61–70, Lisbon, Portugal, Sep 2003. Springer-Verlag LNCS.

[3] M. Sima, S. Vassiliadis, S.Cotofana, J.T. van Eijndhoven, and K. Vissers. Field-Programmable Custom Computing Machines - A Taxonomy. In *FPL*, volume 2438, pages 79–88, Montpellier, France, Sep 2002. Springer-Verlag LNCS.

[4] S. Vassiliadis, S. Wong, and S. Cotofana. The MOLEN $\rho\mu$-Coded Processor. In *FPL*, volume 2147, pages 275–285, Belfast, UK, Aug 2001. Springer-Verlag LNCS.

Op Type	unsigned	signed
char	rlwinm r1,0,0xff → r4 rlwinm r2,0,0xff → r5 mullw r4, r5 →r3	
short	rlwinm r1,0,0xffff → r4 rlwinm r2,0,0xffff → r5 mullw r4, r5 →r3	extsh r1 → r4 extsh r2 → r5 mullw r4, r5 →r3
int	mulhwu r1, r2 → r3' mullw r1, r2 → r3"	mulhw r1, r2 → r3' mullw r1, r2 → r3"
long long	mullw r2", r1" → r3"" mulhwu r2", r1" → r3"' mullw r2", r1' → r10 mulhwu r2", r1' → r3" mullw r2', r1" → r12 mulhwu r2', r1" → r11 mullw r2', r1' → r13 mulhwu r2', r1' → r3' addc r3"', r10 → r3"' adde r3", r10 → r3" addze r3' → r3' addc r3"', r12 → r3"' adde r3", r13 → r3" addze r3' → r3'	mullw r2", r1" → r3"" mulhw r2", r1" → r3"' mullw r2", r1' → r10 mulhw r2", r1' → r3" mullw r2', r1" → r12 mulhw r2', r1" → r11 mullw r2', r1' → r13 mulhw r2', r1' → r3' addc r3"', r10 → r3"' adde r3", r10 → r3" addze r3' → r3' addc r3"', r12 → r3"' adde r3", r13 → r3" addze r3' → r3'

TABLE II
LIR TRANSLATION OF MIR INSTRUCTION *mul r1, r2 → r3*

[5] J.M. P. Cardoso and H. C. Neto. Compilation for FPGA-based Reconfigurable Hardware. *IEEE Design & Test of Computers*, 20(2):65–75, March/April 2003.

[6] M.B. Gokhale and J.M. Stone. Napa C: Compiling for a Hybrid RISC/FPGA Architecture. In *Proc. IEEE Symp. on Field-Programmable Custom Computing Machines*, pages 126–137, Napa, California, April 1998.

[7] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins. DRESC: A Retargetable Compiler for Coarse-Grained Reconfigurable Architectures. In *FPL*, pages 166–173, Hong Kong, Dec 2002. Springer-Verlag LNCS.

[8] A. La Rosa, L. Lavagno, and C. Passerone. Hardware/Software Design Space Exploration for a Reconfigurable Processor. In *Proc. of the DATE 2003*, pages 570–575, 2003.

[9] Z.a. Ye, N. Shenoy, and P. Banerjee. A C Compiler for a Processor with a Reconfigurable Functional Unit. In *ACM/SIGDA Symposium on FPGAs*, pages 95–100, Montery, California, USA, 2000.

[10] S. Vassiliadis, G. N. Gaydadjiev, K. Bertels, and E. Moscu Panainte. The Molen Programming Paradigm. In *Proc. of the Third International Workshop on Systems, Architectures, Modeling, and Simulation*, pages 1–7, Samos, Greece, July 2003.

[11] G.K. Kuzmanov and Stamatis Vassiliadis. Arbitrating Instructions in an -coded CCM. In *FPL 2003*, volume 2778, pages 81–90, Lisbon, Portugal, Sep 2003. Springer-Verlag LNCS.

[12] E. Moscu Panainte, K. Bertels, and S. Vassiliadis. Compiling for the Molen Programming Paradigm. In *FPL 2003*, volume 2778, pages 900–910, Lisbon, Portugal, Sep 2003. Springer-Verlag LNCS.

[13] http://suif.stanford.edu/suif/suif2.

[14] http://www.eecs.hardward.edu/hube/research/machsuif.html.

[15] S. Sobek and K. Burke. *PowerPC Embedded Application Binary Interface 32-Bit Implementation, Version 1.0*.