Dear Bernhard,

This is a summary of work done at University of Brest for mapping Smalltalk computation

to hardware.

Dear Bernhard,

This is a summary of work done at University of Brest for mapping Smalltalk computation to hardware.

1) efficiency is concurrency in execution

During decades researchers and industry was running behind the work of Xerox PARC on micro-programmed computers, such as the Dorado. The Dorado implemented a pipeline for decoding languages intermediate code developped for this purpose.

Smalltalk bytecodes as delivered for Smalltalk-80 were of this kind.

Behind the streaming engine that prepared the execution, there was the micro-machine routines translated by PARC into micro-program control executing several internal operations in parallel. The blue book implementation part is just a description of how the instructions are executed.

(For those who do not know about micro-programming, there are primers written in the 70's by Michael Flynn to explain link between the instruction set, micro-program structure, and program execution. It takes 1 day to read, understand, and experiment: look at Ci lines that activate concurrent operations from a micro-code word.)

[Flynn] Micro-programming, includes Rosin machine, in Computer Architecture 1972. Annex 1.

[Dorado] Retrospective, Pier.

http://bitsavers.trailing-edge.com/pdf/xerox/parc/techReports/ISL-83-1_A_Retrospective_on_the_Dorado_A_High-Performance_Personal_Computer.pdf

So, Smalltalk was fast enough in 1980, it was well understood for a machine execution, and became so slow when executed on micro processors, despite valuable efforts at UCBerkeley, Stanford and others.

2) concurrency is now easily available

Probably one could implement Rosin machine in few days, plus a compiler to translate a calculator syntax into instruction set for it. Research could rather examine chances to execute high level computer language at the current level of resources as FPGAs devices and shared memory accelerators. In the second case, one can recall that previous generation of massive parallel computers used Smalltalk to model execution. It was the case for Maspar machine which debugger was showing clearly a ParcPlace support to display assertions on parallel (plural) variables.

This debugger inspired some work to support parallel execution on GPU as shown in [VanLong].

[VanLong] Master report. Annex 2. http://wsn.univ-brest.fr/pottier/long.pdf
(See also https://www.mdpi.com/1424-8220/18/7/2323 about parallel physical simulation)

3) execution and definition of object spaces can replace types.

Smalltalk does not have types. Types ease program checking and preparation of execution on machines. Instead, Smalltalk provides classes to resolve call to methods. Instead of relying on class tags, we can also rely on object sets (called contexts), solve binding on these object sets, and produce logic networks that will replace the slow interpretation or execution by one call to a dedicated function block.

Computer operative parts are either logic networks, combinational or sequential, defined at boolean level. Thus this schema is the common way to produce operative blocks, even dynamically. The first stage is a high level look-up table tree production organized around method calls, with evolved backtracking memory optimizers. This could be suitable for architectures such as GPUs or memory based nano-architectures. The second stage is mapping to reconfigurable architectures, expecially those oriented to logic, such as FPGAs.

The Madeo framework address Smalltalk to technology translation in its different stages address this question, such as:

[FCCM96] Smalltalk blocks to logic,

4th IEEE Symposium on FPGAs for Custom Computing Machines (FCCM '96), Napa Valley, CA, USA, April 17-19, 1996. IEEE 1996, ISBN 0-8186-7548-9
also at http://wsn.univ-brest.fr/reports/96/st80-2-FPGA.ps.Z

[SAMOS2002] A LUT based approach for high level synthesis,
Shuvra S. Bhattacharyya, E.F. Deprettere, Jurgen Teich, (Eds.), Domain-Specific Processors:

Systems, Architectures, Modeling, and Simulation, Hardcover, pp. 261, plus XV, Marcel Dekker, Inc., New York, 2004, ISBN 0-8427-4711-9

also at http://wsn.univ-brest.fr/reports/2002/samos2.pdf

4) Mapping LUT systems to hardware were addressed by two excellent PhD works, one that draws a flattened LUT tree on an FPGA, the other one that address nano technologies, largely connected to Pr Moritz work at Umass.

As a conclusion (I hope it is not final), there are large opportunities to improve specific execution speed of Smalltalk engines using new 'back to the future' approaches. We just need to stop doing bad copies of the sacred 40 years implementation, pretending that it is a 'virtual machine' for today, look to GPUs, reconfigurable devices, and new technologies especially those based on low energy and fast physical memories.

Bernard Pottier
Université de Brest, France

pottier@univ-brest.fr

Annex 1 : Rosin's micro-programmed pedagogical machine

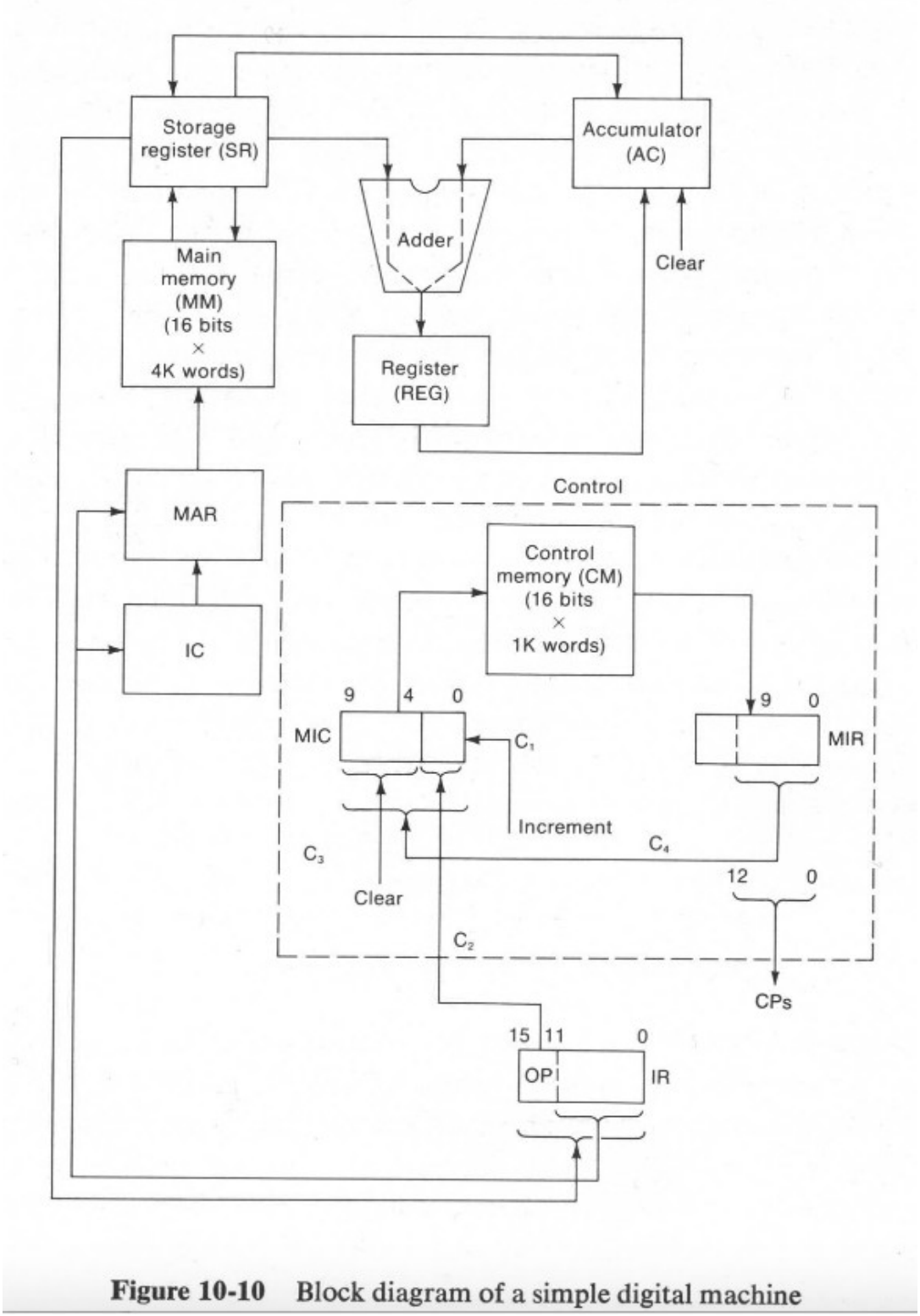

Figure 10-10 Block diagram of a simple digital machine

DjVu PostScript document

Chapitre 10 : Interprétation, microprogrammation, et le controle d'un ordinateur(Michael Flynn, 40 pages)

Annex 2

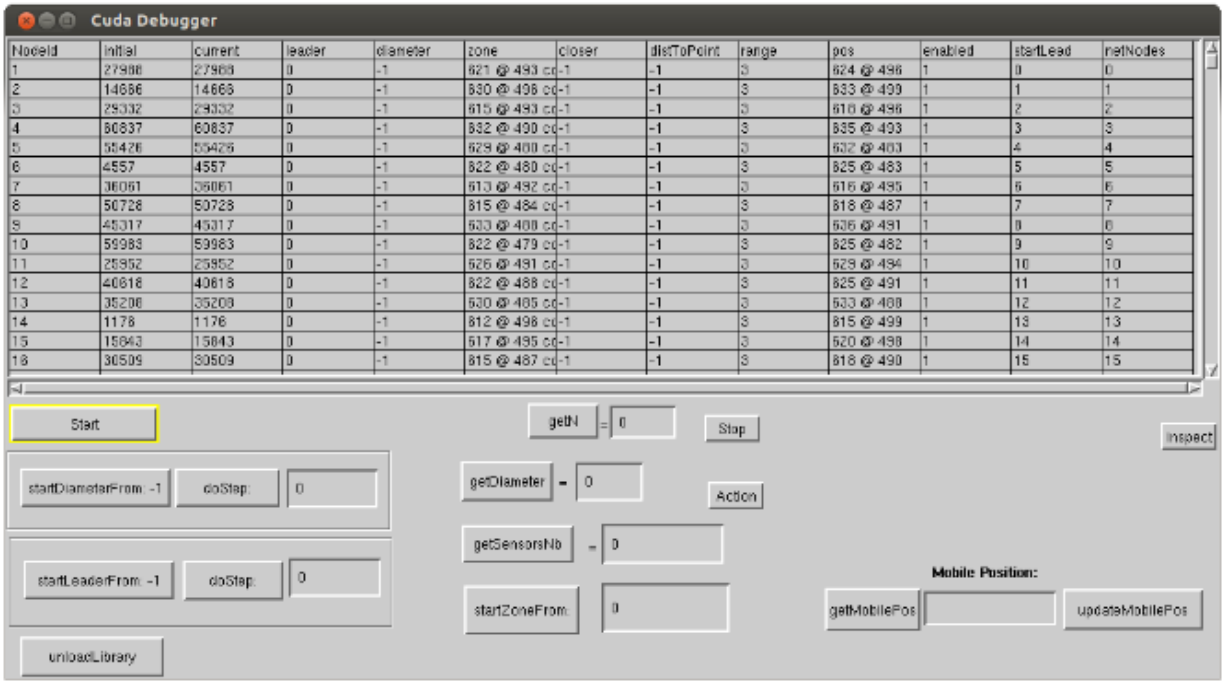

Figure 4.7: Debugger interface at high level control

This view is a browser implemented on data set up in GPU memory. As all these data structures are produced from Smalltalk, compiled into CUDA code using nvcc, they can be written and read using a DLL, from the master Visualworks engine. Thus it becomes possible do build step by step application level debugger enabling developpers to follow remote execution on GPU (few thousand of processors and larger number of application nodes.

This was implemented in NetGen. It can support arbitrary networks, such as wireless sensor nodes, with moving nodes, or, other graphs for other usages.