



SMIPS Multimedia Extension

Group 2

Myron King

Asif Khan



Motivation & Utility

Motivation: Do we really need a Multimedia Extension at all?

- Intel's success with MMX and SSE
- The entire GPU industry (ATI, Nvidia, Intel)
- The nascent PPU industry (Ageia, Sony)
- MIPS MDMX from SGI
- Sony's in-house GPU (PSP, PS3)
- Only barrier to ubiquity is how to compile to them!

Utility: What does a Multimedia Extension look like and what does it do?

- Expose vector primitives (vector registers replace scalar ones)
- Expose DWORD primitives within each vector
- Add opcodes which are useful for target applications
- Make claims about memory interaction
- Convince others it's actually useful!



Getting Started

Nothing new under the sun: why reinvent the wheel?

- Interesting work; lots of infrastructure already in place
- Until you implement something, you don't fully "grok" it
- Still an active area in research, both industrial and academic
- Cross-pollination which took place in exploration could lead to interesting projects in the future
- Asif is tenacious Bluespec hacker and does the heavy lifting!

Coming up with the specifics:

- DirectX Shader Language (vertex shaders especially)
- MMX and SSE for instruction set extension
- Discussions with Chris Batten (exploration)
- Arvind's insistence on specifying the micro-protocol details early on led us to an implementation which would ensure SC but with minimal interlocking (for greater efficiency)



Changing smipsv2

Adding the Coprocessor:

- At first all in one module but onerous compile times as well as good design practice forced us to modularize our design
- Definition of interfaces for transfer of Data (and state) from control processor to coprocessor
- Once we gained adequate Bluespec skills, this came quite naturally (getting over the learning curve, easier said than done)

Implementing the Instructions:

- Determining which instructions run on which processor (some on both) was the first step.
- Some Cop2 instructions must be run on the control processor as well (SC follows naturally if done correctly)
- Restrictions on Cop2 instructions allow for easier implementation (no CF instructions and no non-aligned loads and stores)



Changing smipsv2

What did we do to SMIPSV2:

- Add a coprocessor module with some new opcodes.
- Add a new rule “dispatch” between “pcGen” and “exec”
- Change the memory caches: enlarge cache lines to support 128 bit loads and stores
- Add more control logic for the interaction with the control processor
- Add some cop2 instructions to the control processor execution (those which need both)

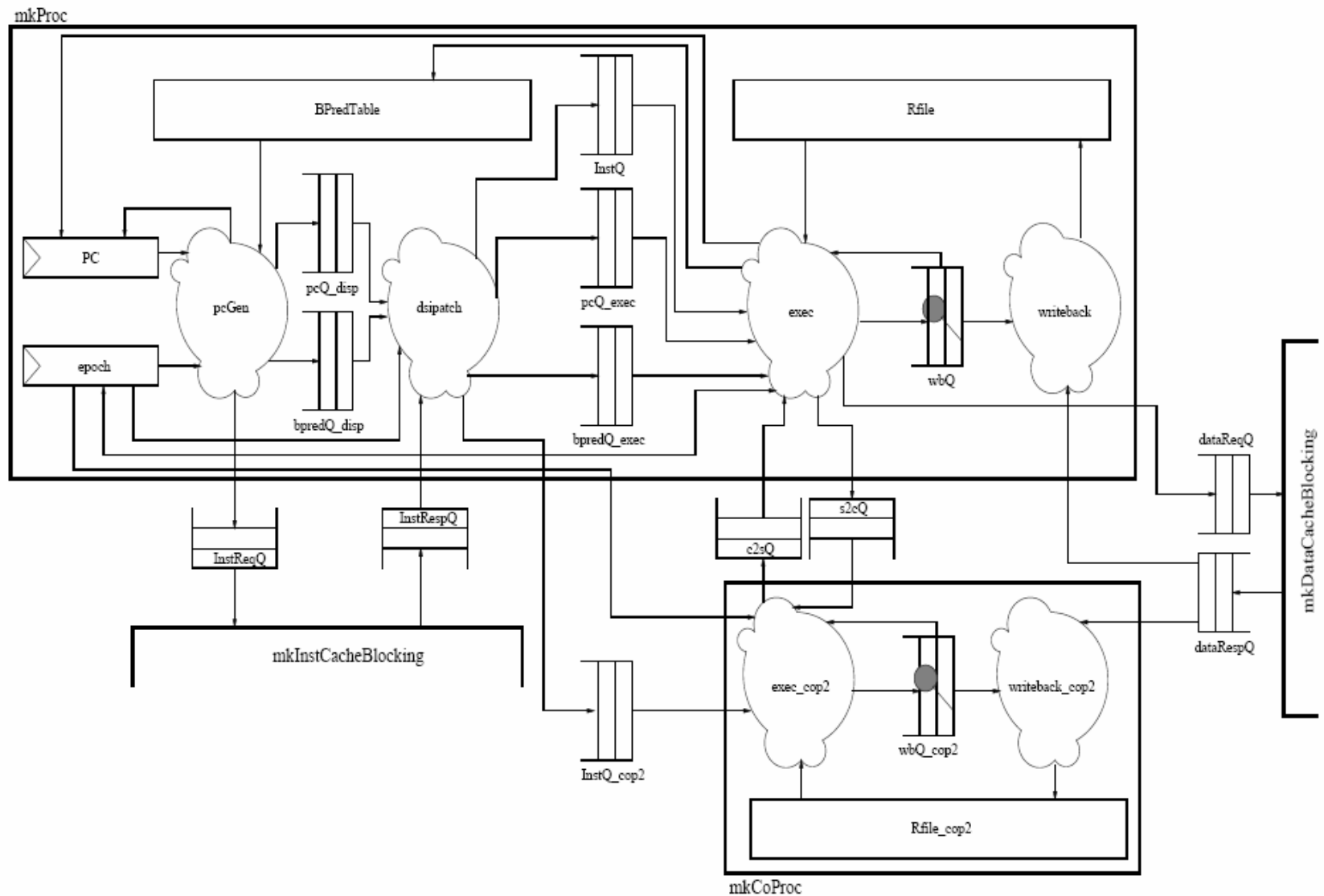
What's in the Coprocessor:

- only execution and write back stages
- Cache interface needed to be changed to route responses
- lots of gotcha's!

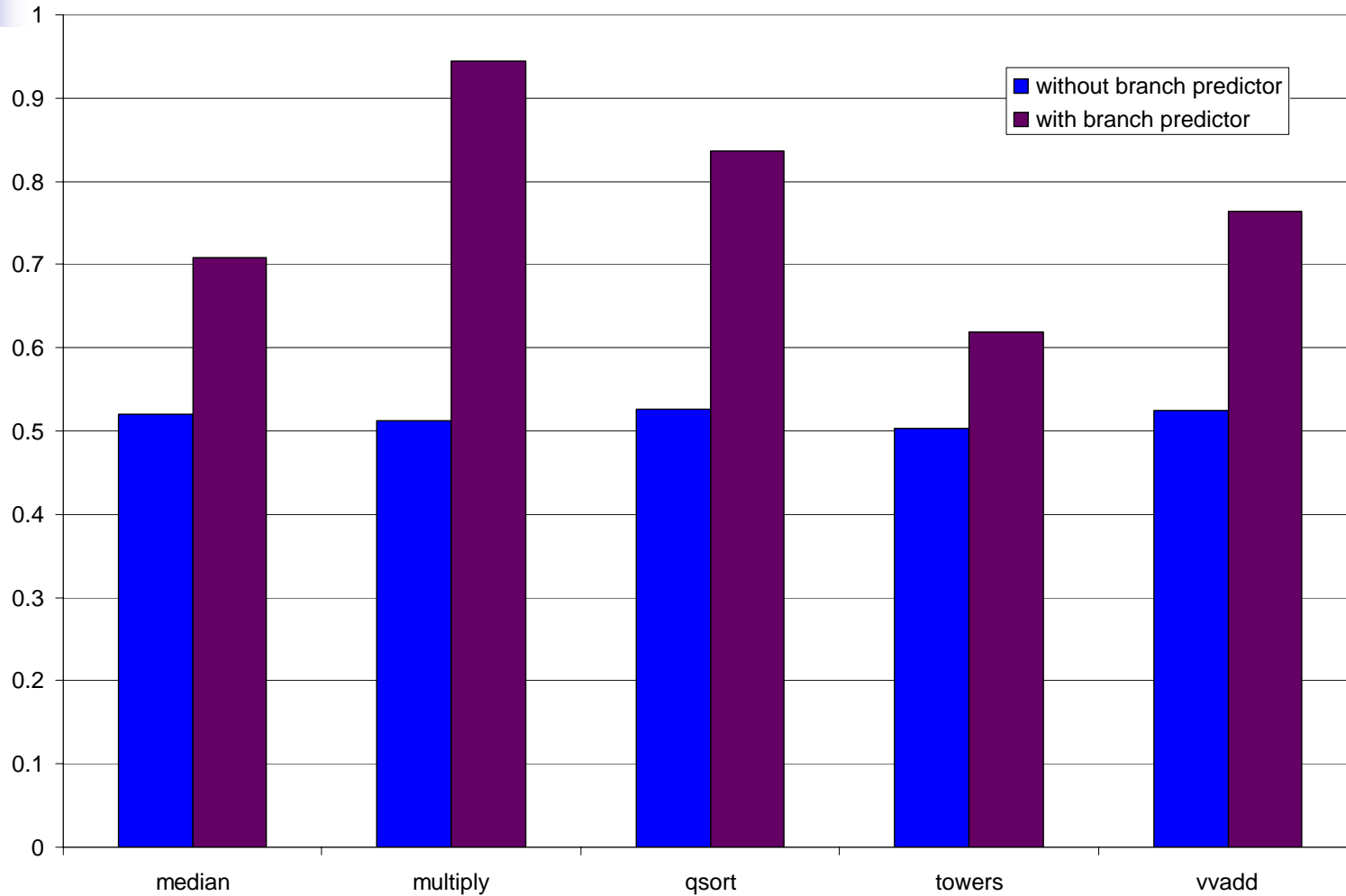
Getting everything up and running:

- Add pre-asm.pl to tool path
- Write tests and benchmarks (hand-writing assembly code is no fun!)

Microarchitecture

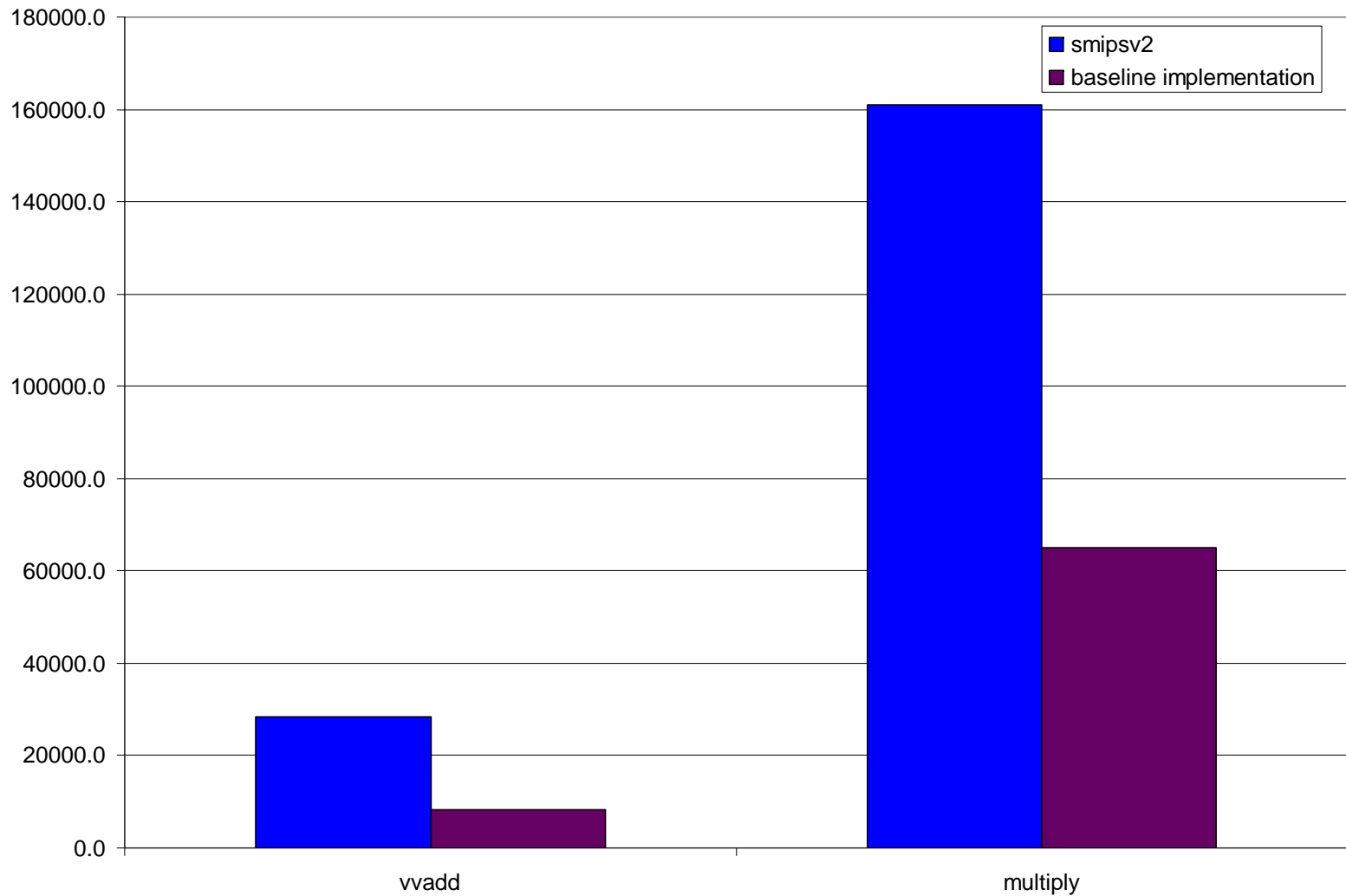


IPC's for Various Benchmarks on smipsv2



Branch prediction works: but you already knew that!

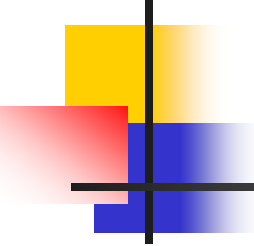
Runtime Comparison between smipsv2 and Baseline Implementation





Exploration 1: *16-DWORD* *Vectors*

- 16-dword vectors but still 4 lanes in the coprocessor
- Register File enlarged to 24 4-dword registers from 8 4-dword registers
- Semantics of the control processor instructions and the data transfer instructions remain unaltered
- Exec rule changed in the control processor to execute LWC2 and SWC2 in 4 cycles
- Exec rule changed in the coprocessor to execute all instructions other than the data transfer instructions in 4 cycles
- Writeback rules in both the control processor and the coprocessor remain unaltered

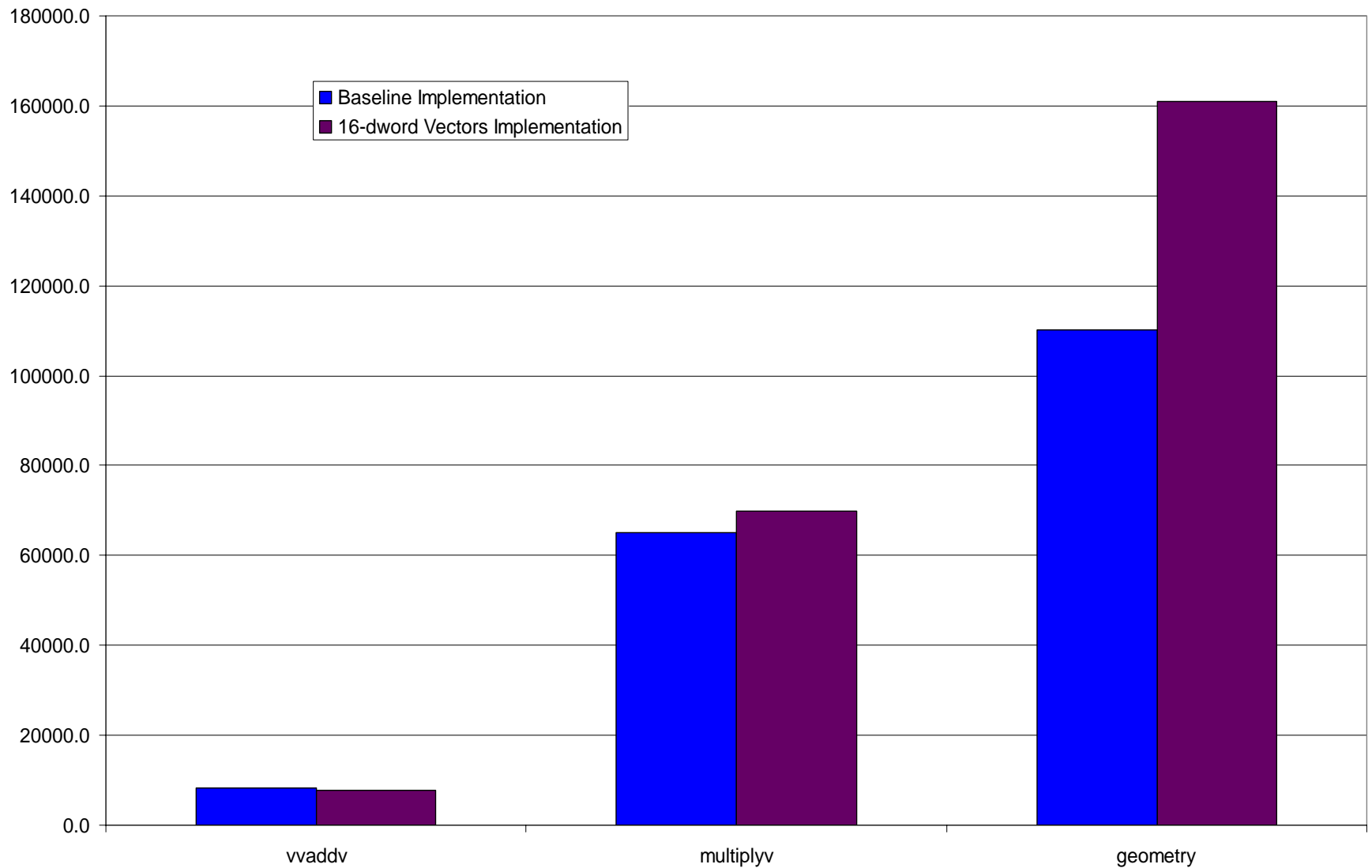


Exploration 1: *16-DWORD* *Vectors*

Discarding Mispredicted Branches

- Single epoch register scheme from smipsv2 falls apart
- Coprocessor takes multiple cycles to execute each instruction, allowing the control processor to run ahead
- Another epoch register added which is incremented every time a branch instruction is dispatched
- All the coprocessor instructions are dual-tagged
- Extra checks in the exec rule of the coprocessor to make sure that all instructions which were dispatched before the branch instruction get executed

Runtime Comparison between Baseline and Exploration 1

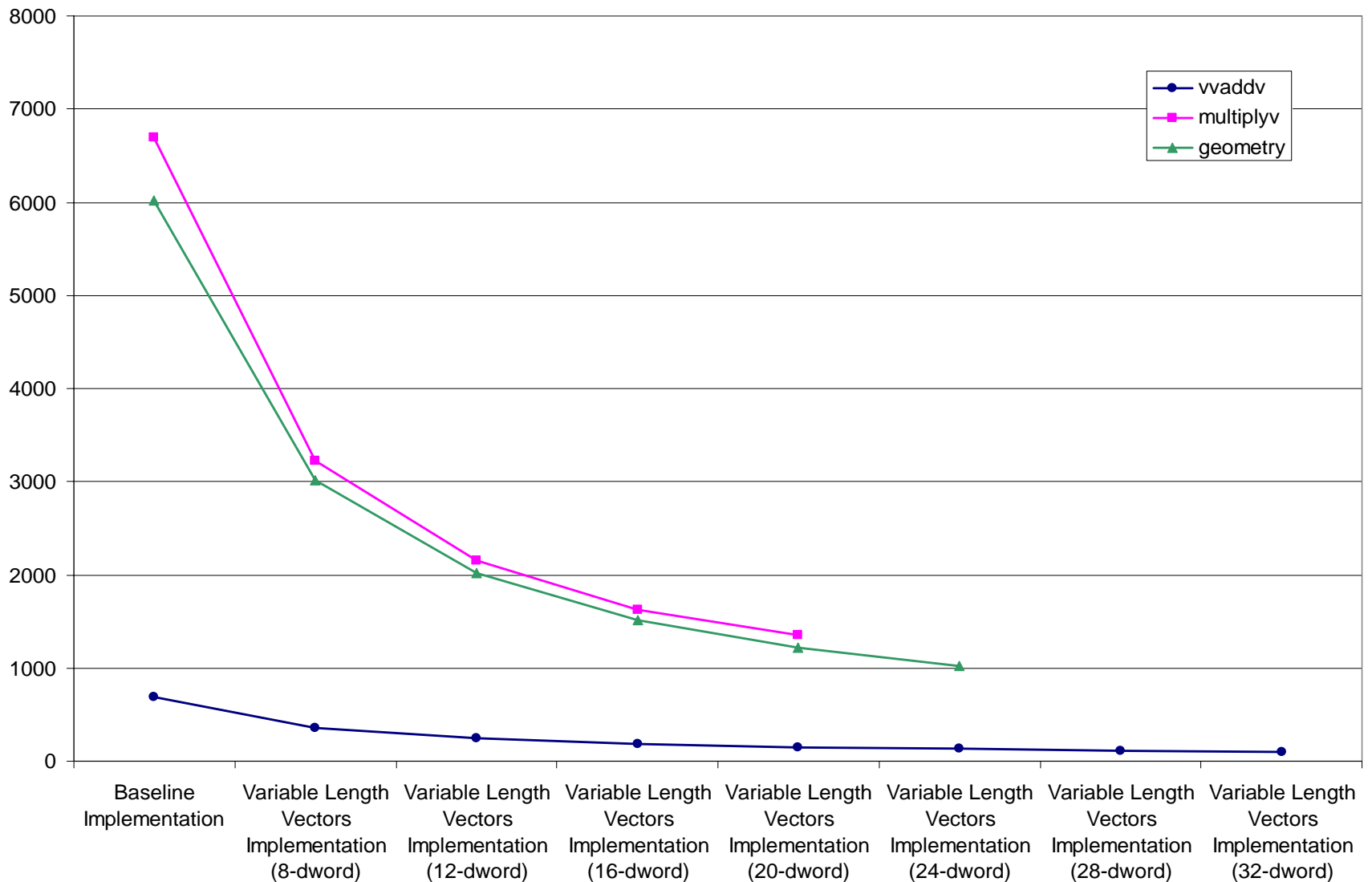




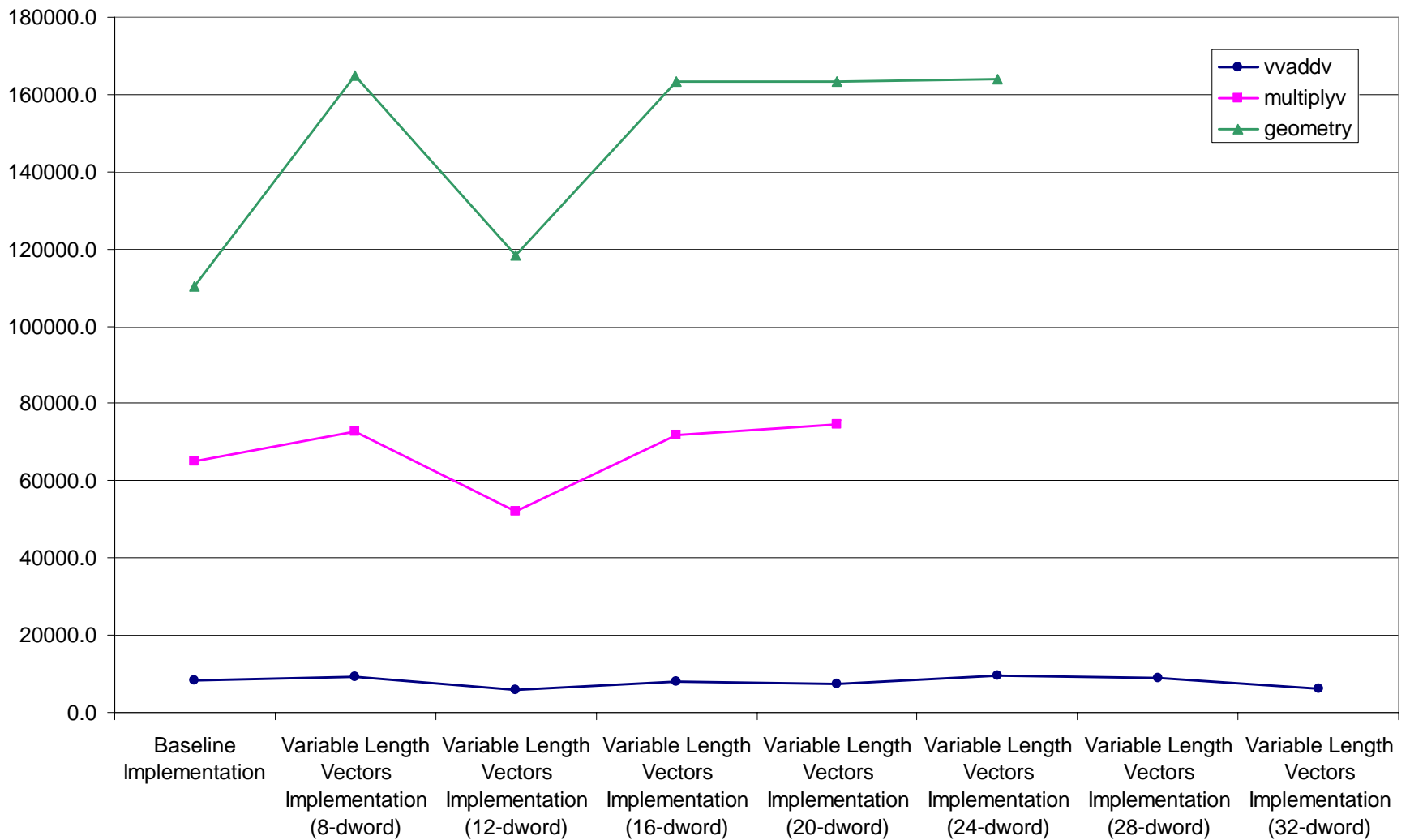
Exploration 2: *Variable Length Vectors*

- A control register is added which allows the programmer to set the length of the vector registers using the CTC2 instruction
- Length has to be a multiple of 4 and maximum length restricted to 32-dwords
- Register File further enlarged to 32 4-dword registers
- Mask bits increased to 32
- Changes to the exec rules in the control processor and the coprocessor similar to exploration1

Number of Instructions for Custom Benchmarks



Runtimes for Custom Benchmarks (ns)



Exploration 3: *ALU changes for clock speed*

improvement

- The dot4 instruction creates the longest combinational path
- dot4 instruction broken down into mulv and addh instructions
- Register File size is the same as that for the baseline implementation
- Minor changes in design to accommodate for the added addh instruction



Timing and Area Comparison

Total Area and Effective Clock Period of Different Implementations from the Synthesis Tool		
	Area (units)	Effective Clock Period (ns)
smipsv2 Implementation	28,837.25	4.26
Baseline Implementation	87,413.50	5.50
16-dword Vectors Implementation	147,652.25	5.50
Variable Length Vectors Implementation	172,251.00	5.84
Alternate ALU Implementation	104,651.75	5.00

Total Area and Effective Clock Period of Different Implementations from Encounter		
	Area (sq micron)	Effective Clock Period (ns)
smipsv2 Implementation	464,849.30	7.174
Baseline Implementation	1,415,025.90	9.453
16-dword Vectors Implementation	2,466,711.70	14.520
Variable Length Vectors Implementation	2,799,400.60	14.889
Alternate ALU Implementation	1,708,809.50	10.782



Conclusion

- The baseline implementation is a win!
- Explorations have not proven very fruitful
- Memory bottleneck with lengthened vectors
- Not changing the register file size increases register pressure on benchmarks
- Needed more time for floor planning to get better timing and area from Encounter
- A few more benchmarks perhaps
- We're happy with what we've accomplished



Thank you
