A Hybrid-FPGA System for On-Board Data Processing Targeting the MATMOS FTIR Instrument

Dmitriy L. Bekker (JPL/RIT), Marcin Lukowiak (RIT), Muhammad Shaaban (RIT), Jean-Francois L. Blavier (JPL), Paula J. Pingree (JPL)

March 5, 2008
IEEE Aerospace Conference
Big Sky, MT

Jet Propulsion Laboratory (JPL)
California Institute of Technology

Department of Computer Engineering
Rochester Institute of Technology (RIT)

Outline

- Definition of terms
- Motivation
- FTIR Spectrometry
- Project overview
- Performance evaluation of system builds
  - Base system with SW optimizations
  - APU-FPU system
  - APU-FPU system with dot product co-processor
- Results summary
- Recommendation for future work

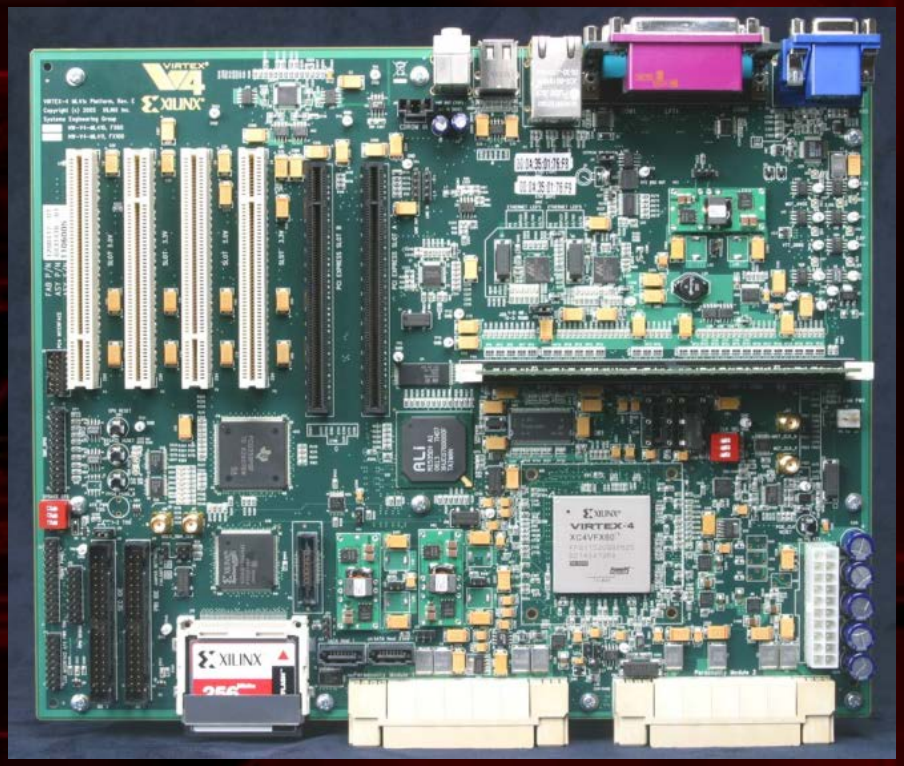

Figure 1: Xilinx ML410 development board

Definition of Terms

- APU — Auxiliary Processor Unit (connects to co-processors)
- BRAM — Block RAM (on Xilinx FPGAs)
- DP — Double-precision floating-point
- F2C — FORTRAN-to-C Converter (automatic)
- FCB — Fabric Co-processor Bus (for direct link to APU co-processor)
- FTIR — Fourier Transform Infrared Spectrometry
- FPGA — Field Programmable Gate Array (reconfigurable logic)
- FPU — Floating Point Unit
- Hybrid-FPGA — An FPGA with embedded processor core(s)
- MARVEL — Mars Volcanic Emission and Life (mission)
- MATMOS — Mars Atmospheric Trace Molecule Spectroscopy (instrument)
- OPB — On-chip Peripheral Bus (low speed)
- Perflib — IBM Performance Libraries (SW optimization)
- PLB — Processor Local Bus (high speed)
- PPC405 — PowerPC 405 embedded processor inside V4FX FPGA
- RAD750 — PowerPC 750 radiation-hardened SBC (by BAE)
- SBC — Single-board computer
- SP — Single-precision floating-point
- V2P — Xilinx Virtex-II Pro
- V4FX — Xilinx Virtex-4 FX FPGA

Motivation

- Proposed 2011 Mars Scout mission (MARVEL) will study Martian atmosphere (*not selected*)
- Primary instrument is a solar occultation Fourier Transform Spectrometer, MATMOS
  - Measures infra-red spectrum of direct sunlight
  - Produces large volumes of floating-point data
  - Data must be processed in orbit prior to downlink
  - Requires two BAE RAD750 single-board computers (SBC)
- All-software implementation on Virtex-II Pro (V2P) FPGA incapable to meet processing needs
- V4FX has potential to meet processing needs with mixed HW/SW system

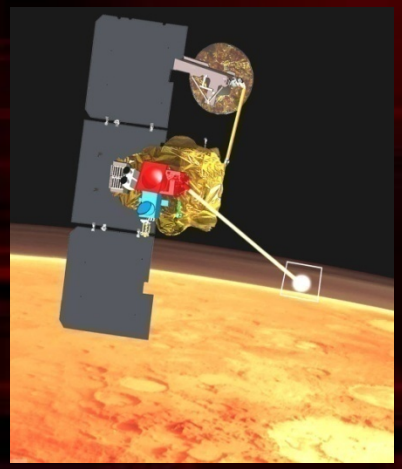

Figure 2: MARVEL spacecraft

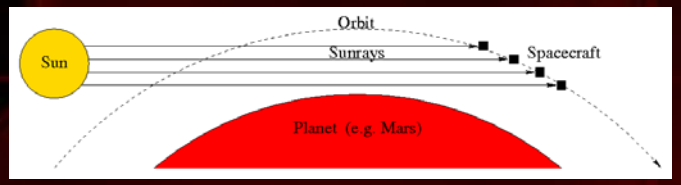

Figure 3: One observation (sunset)

FTIR Spectrometry Data Collection

- MATMOS measures 850-4300 cm^{-1} region of infra-red spectrum
- High 0.02 cm^{-1} resolution
- Records 26 spectra per occultation, 52 per orbit
- Duration of occultation: 78-169 sec
- Each spectrum must be collected in 3.0-6.5 sec
- Each spectrum has 172,500 spectral elements

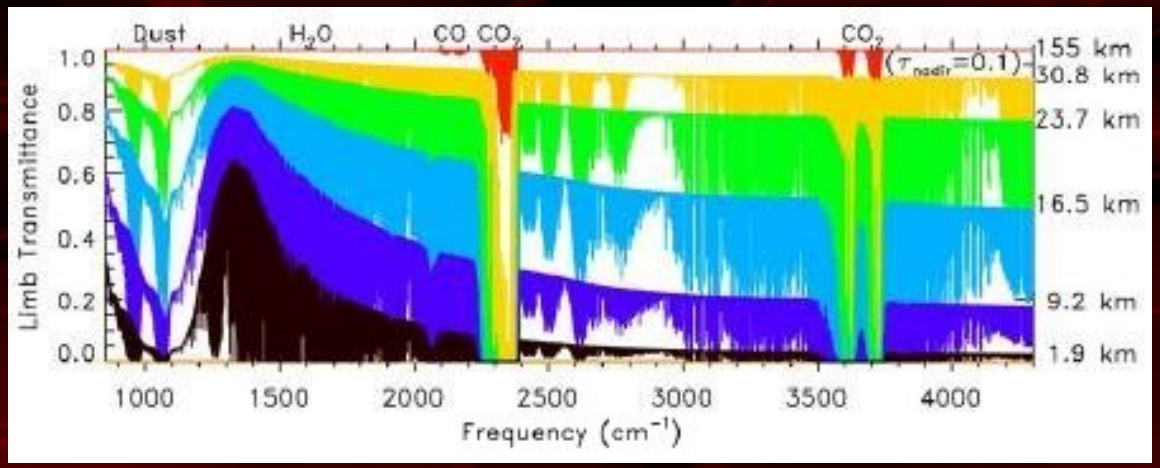

Figure 4: Sample spectrum under low dust conditions

FTIR Spectrometry Data Processing Steps

- Re-sampling
  - Frequency modulation in time-domain caused by variations of the mirror velocity
  - Re-sample to path-difference domain (removes modulation)
- Phase correction
  - Using convolution
  - Makes interferogram symmetrical about zero path difference
- Compute spectrum
  - Using FFT
  - Produces output with smaller dynamic range (32 bits → 16 bits)
  - Reduce spectral range
- Average scans taken above atmosphere
- Lossless compression

Data Processing Step	Reduction Factor	Data Size (Mbytes)
Raw Interferogram	- - -	659.18
Interferogram Re-sampling	1.83	360.21
Phase Correction	2.00	180.10
Fast Fourier Transformation	6.10	29.53
Spectra Averaging	2.00	14.76
Lossless Compression	1.80	8.20
Net Data Reduction	**80.37**	**8.20**

Table 1: Reduction in data volume due to on-board data processing

Project Overview

- Port existing floating-point intensive FTIR spectrometry algorithm into PowerPC 405 (PPC405) embedded processor
- Apply software optimizations
  - Use single-precision math library
  - Use IBM *Perflib*
- Identify bottlenecks
- Apply hardware optimizations
  - Xilinx single-precision APU-FPU
  - Develop single-precision dot-product co-processor
- Evaluate computational performance (execution time)

FTIR Base System Hardware

- 100 MHz PLB
  - DDR2 memory controller (for 512 MB DIMM)
  - Bridge to OPB
- 100 MHz OPB
  - RS232 UART for standard I/O
  - SystemACE compact flash (CF) controller (for storage of raw interferogram data)
- 200 MHz PPC405
  - Caches on
  - Burst and/or cacheline transfers enabled
- 13% of slices occupied
- Original SW in FORTRAN
  - Converted to C using F2C
  - Some modification necessary

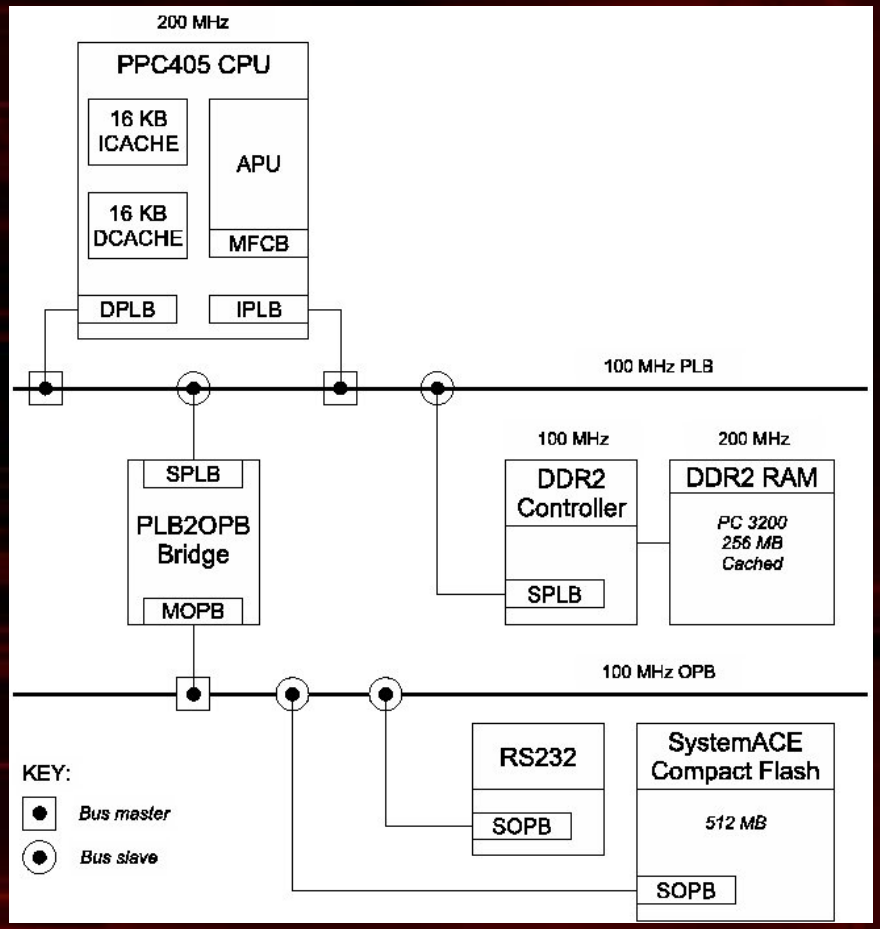

Figure 5: FTIR base system hardware

FTIR Base System Performance

	ML410 (Xilinx)	V2P (NASA JPL)		ML410 (Xilinx)		
PPC405 Freq.	200 MHz	300 MHz		200 MHz		
Interferograms	1	104		scaled to 104		
Detectors	1	2		scaled to 2		
Software Component	**Time (sec)**	**Time (min)**	**Cycles** ($\times 10^{12}$)	**Time (min)**	**Cycles** ($\times 10^{12}$)	**Speedup**
Re-sampling	1197.7938	3404	61.272	4152	49.824	0.82x
Spectrum (Phase Correction, FFT)	117.4963	488+272	13.680	407	4.884	1.87x
TOTAL	1315.2901	4164	74.952	4559	54.708	0.91x
Efficiency Score[5]	-	1.00		1.37		-

Table 2: FTIR base system performance

FTIR Base System Performance

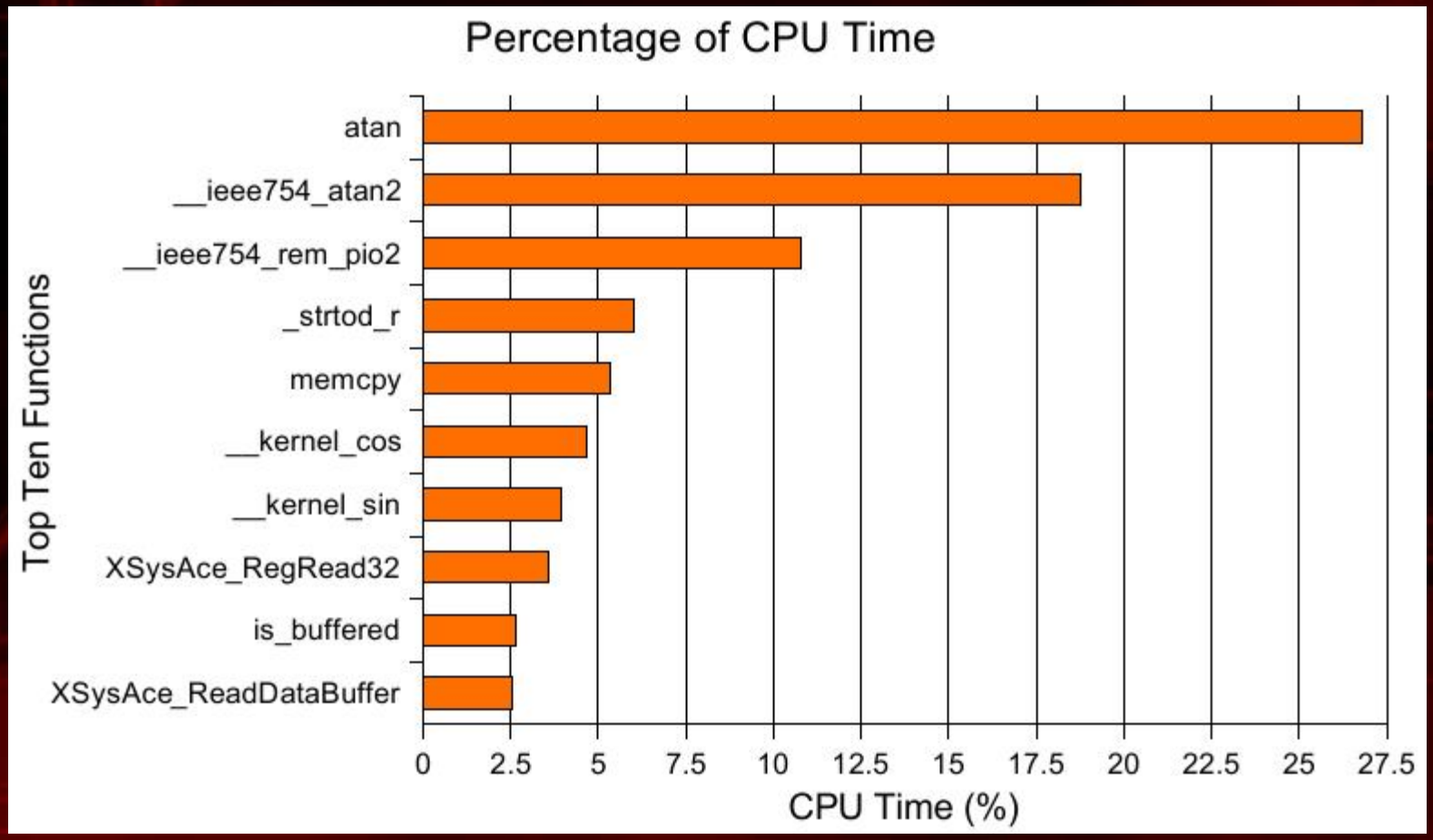

Figure 6: Profiling results for FTIR base system

Software Optimizations
SP-only Math and IBM *Perflib*

- SP-only math
  - Replace DP trig functions with SP trig functions
  - Simple find/replace operation
- IBM *Perflib*
  - Requires no changes to source
  - Achieves higher performance than GCC routines by
    - Avoiding load/use dependencies
    - Avoiding operand dependencies
    - Separating compare and branch
  - Already built in EDK

Routine	Type	Description
fadd	DP	add two DP numbers
fsub	DP	subtract two DP numbers
fmul	DP	multiply two DP numbers
fdiv	DP	divide two DP numbers
fadds	SP	add two SP numbers
fsubs	SP	subtract two SP numbers
fmuls	SP	multiply two SP numbers
fdivs	SP	divide two SP numbers
dtof	conv	convert DP to SP
ftod	conv	convert SP to DP
dtoi	conv	convert DP to INT
ftoi	conv	convert SP to INT
fcmpd	DP	compare two DP numbers
fcmps	SP	compare two SP numbers
fneg	SP/DP	negate a SP or DP number
itod	conv	convert INT to DP
itof	conv	convert INT to SP
ftoui	conv	convert SP to unsign. INT
dtoui	conv	convert DP to unsign. INT

Table 3: Optimized FP routines provided by *Perflib*

Software Optimizations
SP Math & *Perflib*: Performance

SW Optimizations	None (base system)	SP Math Functions IBM Perflib	
PPC405 Freq.	200 MHz	200 MHz	
Interferograms	1	1	
Detectors	1	1	
Software Component	Time (sec)	Time (sec)	Speedup
Re-sampling	1197.7938	244.1836	4.91x
Spectrum (Phase Correction, FFT)	117.4963	44.4243	2.64x
TOTAL	1315.2901	288.6079	4.56x

Table 4: Execution times for system with SP math functions and *Perflib*

Software Optimizations
SP Math & *Perflib*: Performance

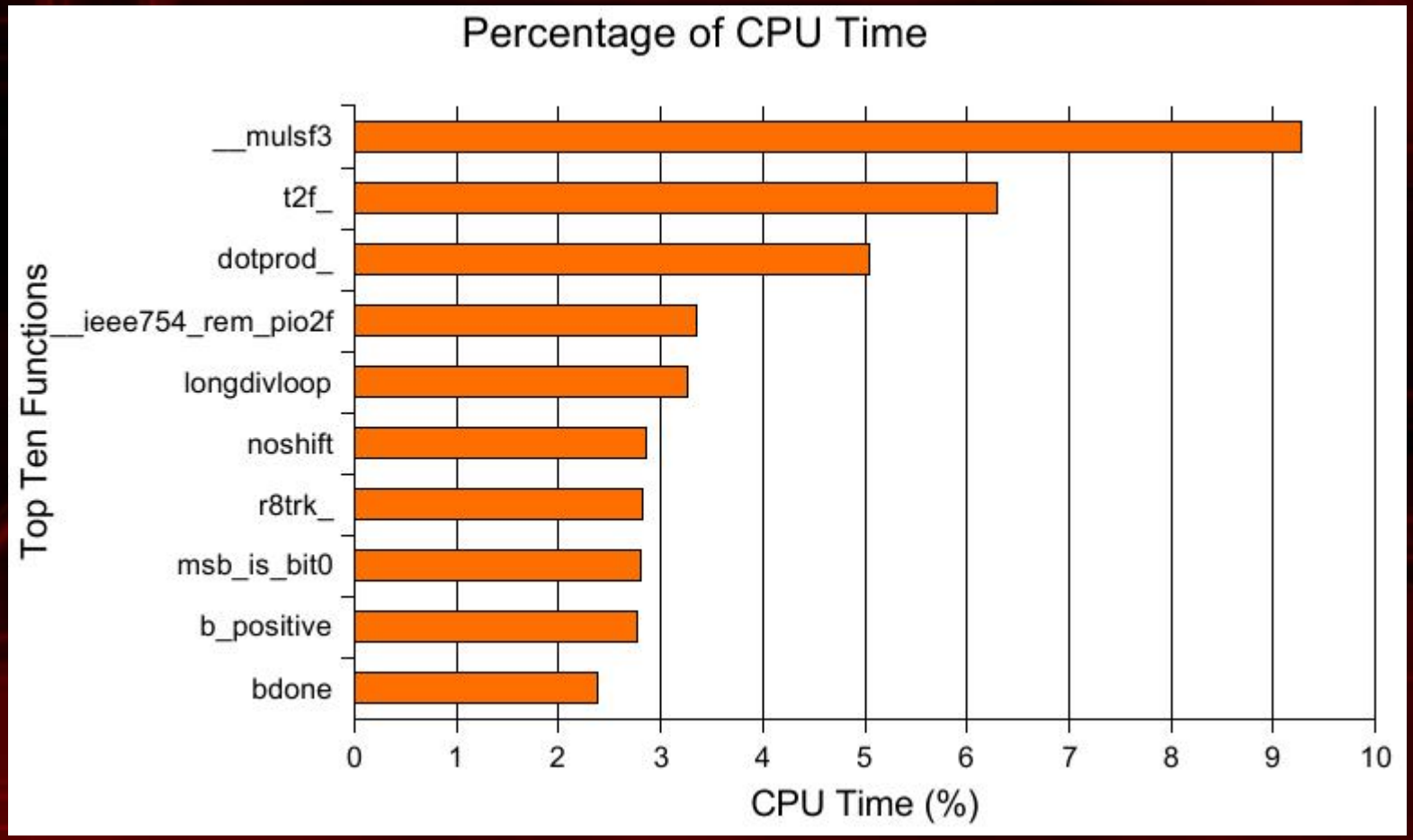

Figure 7: Profiling results for system with SP math functions and *Perflib*

Hardware Optimizations APU-FPU

- All libraries must be recompiled with FPU support enabled (tells system to generate HW FP instructions instead of SW emulation)
- Standard *Perflib* incompatible
  - *Perflib* optimizes string, SP and DP floating-point routines
  - APU-FPU handles SP floating point arithmetic
- Double-precision routines extracted from *Perflib* source (*fadd*, *fsub*, *fmul*, *fdiv*, *fcmpd*)
- HW build
  - FTIR base system + FPU co-processor
  - 19% device utilization

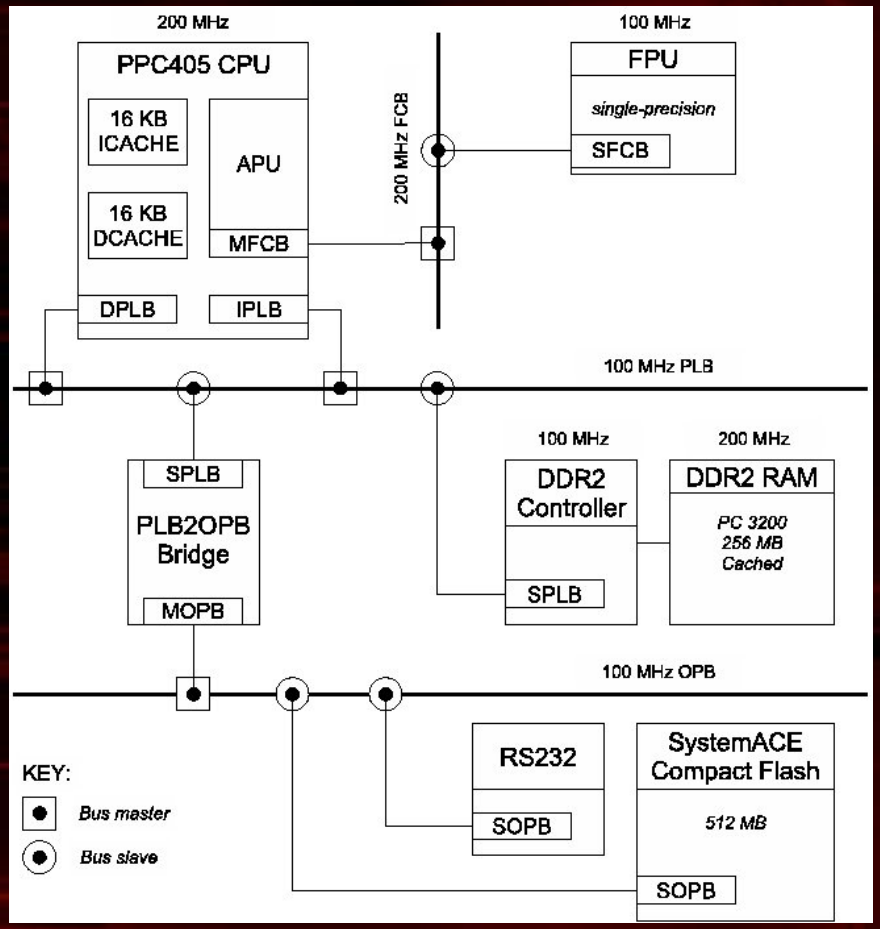

Figure 8: FTIR system with FPU co-processor

Hardware Optimizations
APU-FPU: Performance

SW Optimizations	None (base system)	SP Math Functions IBM Perflib (DP)	
HW Optimizations	None (base system)	APU-FPU (100 MHz)	
PPC405 Freq.	200 MHz	200 MHz	
Interferograms	1	1	
Detectors	1	1	
Software Component	Time (sec)	Time (sec)	Speedup
Re-sampling	1197.7938	151.1817	7.92x
Spectrum (Phase Correction, FFT)	117.4963	14.3672	8.18x
TOTAL	1315.2901	165.5489	7.95x

Table 5: Execution times for system with APU-FPU, SP math functions, and DP *Perflib*

Hardware Optimizations
APU-FPU: Performance

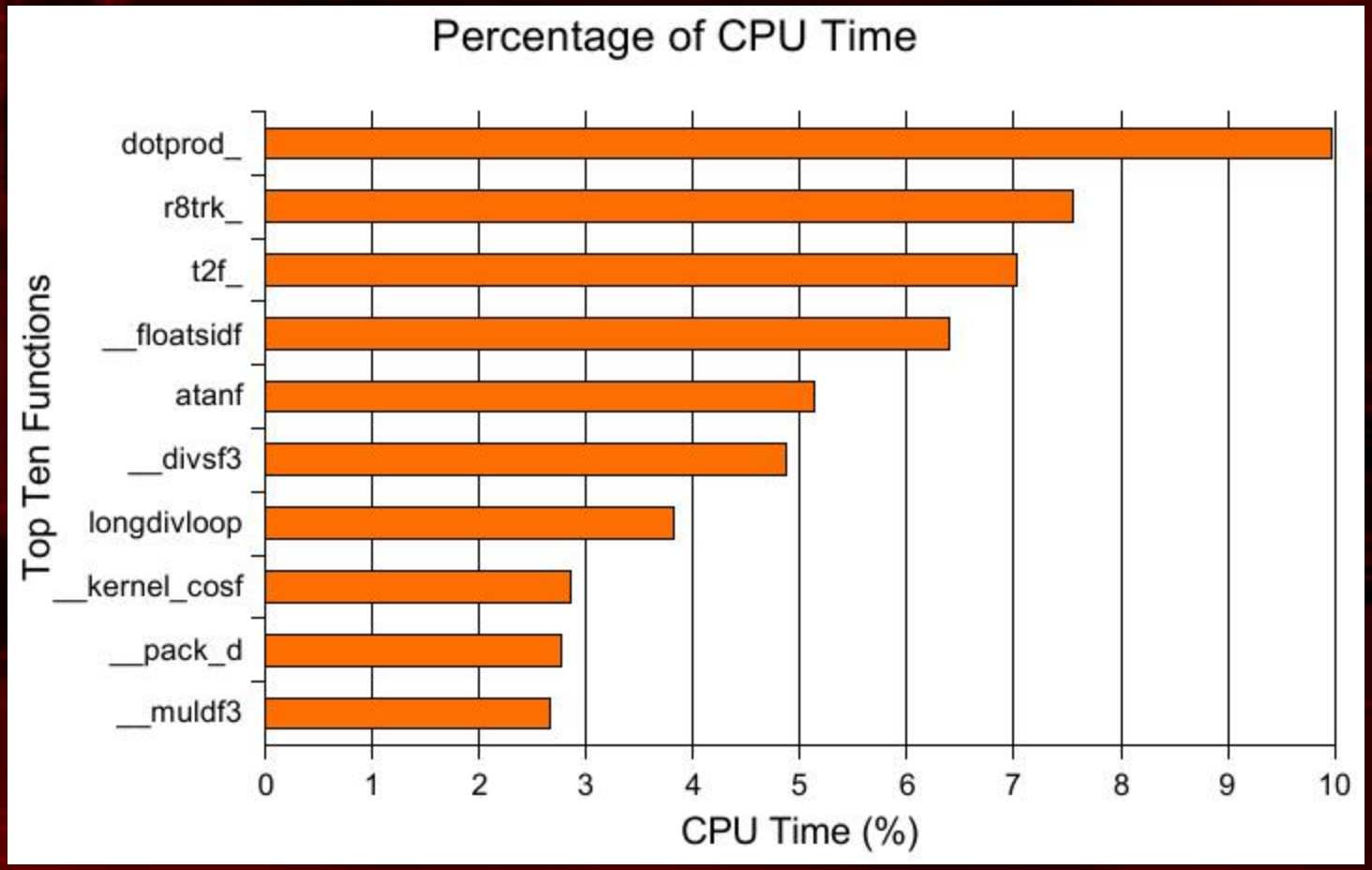

Figure 9: Profiling results for system with APU-FPU, SP math functions, and DP *Perflib*

Hardware Optimizations
Dot-product Co-processor

- Loop iteration
  - *retval = retval + fin[a+] x oper[b+] + fin[c-] x oper[d+]*
- Dot-product computed from both ends of operator array for better precision
- Fits well to APU quadword load / single word store pattern

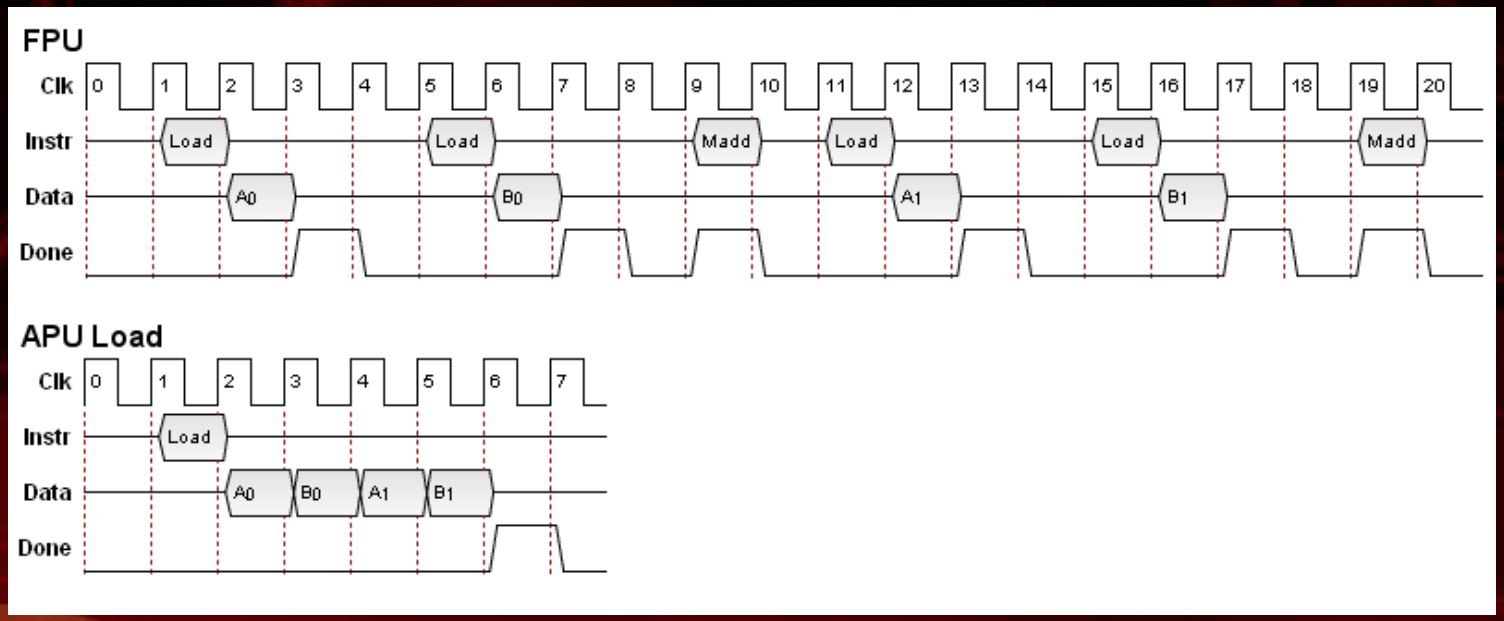

Figure 10: Cycles needed to complete one iteration of dot-product loop

Hardware Optimizations
Co-processor SW Compatibility

- Dot-product function moved to static library
- HW load/store calls integrated
- Poor GCC compatibility
  - Issues with optimization
  - Assembly needs to be hand tuned prior to library compilation

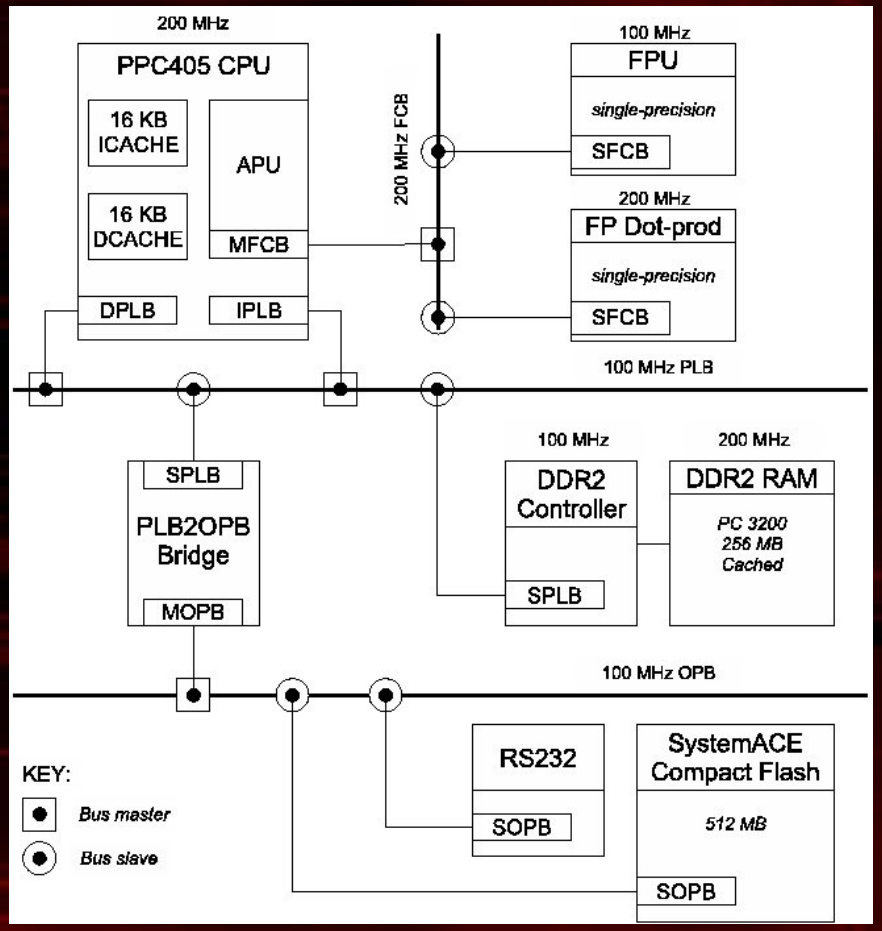

Figure 11: FTIR system with FPU and dot-prod co-processors

Hardware Optimizations
FPU / Dot-prod: Performance

SW Optimizations	None (Base System)	SP Math Functions IBM Perflib		SP Math Functions IBM Perflib	
HW Optimizations	None (Base System)	APU-FPU (100 MHz) Dot-prod (200 MHz)		APU-FPU (100 MHz)	
PPC405 Freq.	200 MHz	200 MHz		200 MHz	
Interferograms	1	1		1	
Detectors	1	1		1	
Software Component	Time (sec)	Time (sec)	Speedup	Time (sec)	Speedup
Re-sampling	1197.7938	149.3297	8.02x	151.1817	7.92x
Spectrum (Phase Correction, FFT)	117.4963	14.3675	8.18x	14.3672	8.18x
TOTAL	1315.2901	163.6972	8.03x	165.5489	7.95x

Table 6: Execution times for system with dot-prod and APU-FPU co-processor, SP math functions, and DP *Perflib*

Hardware Optimizations
FPU / Dot-prod: Performance

- Why so little speedup?
  - Non-contiguous memory access pattern
  - Poor spatial locality
- Testing dot-product co-processor on smaller data set with better spatial locality shows speedup
- Operating on data stored in BRAM results in faster execution over DDR2

CPU (MHz)	FPU (MHz)	Dot-prod (MHz)	Memory	SW-only (sec)	FPU (sec)	Dot-prod (sec)	Speedup
200	-	200	BRAM	0.006500670	-	0.000320910	20.26x
200	-	200	DDR2	0.002594890	-	0.000423610	6.13x
200	100	200	BRAM	-	0.000699365	0.000321390	2.18x
200	100	200	DDR2	-	0.000824490	0.000424330	1.94x

Table 7: Dot-prod core testing with smaller data set

Hardware Optimizations
FPU / Dot-prod: Performance

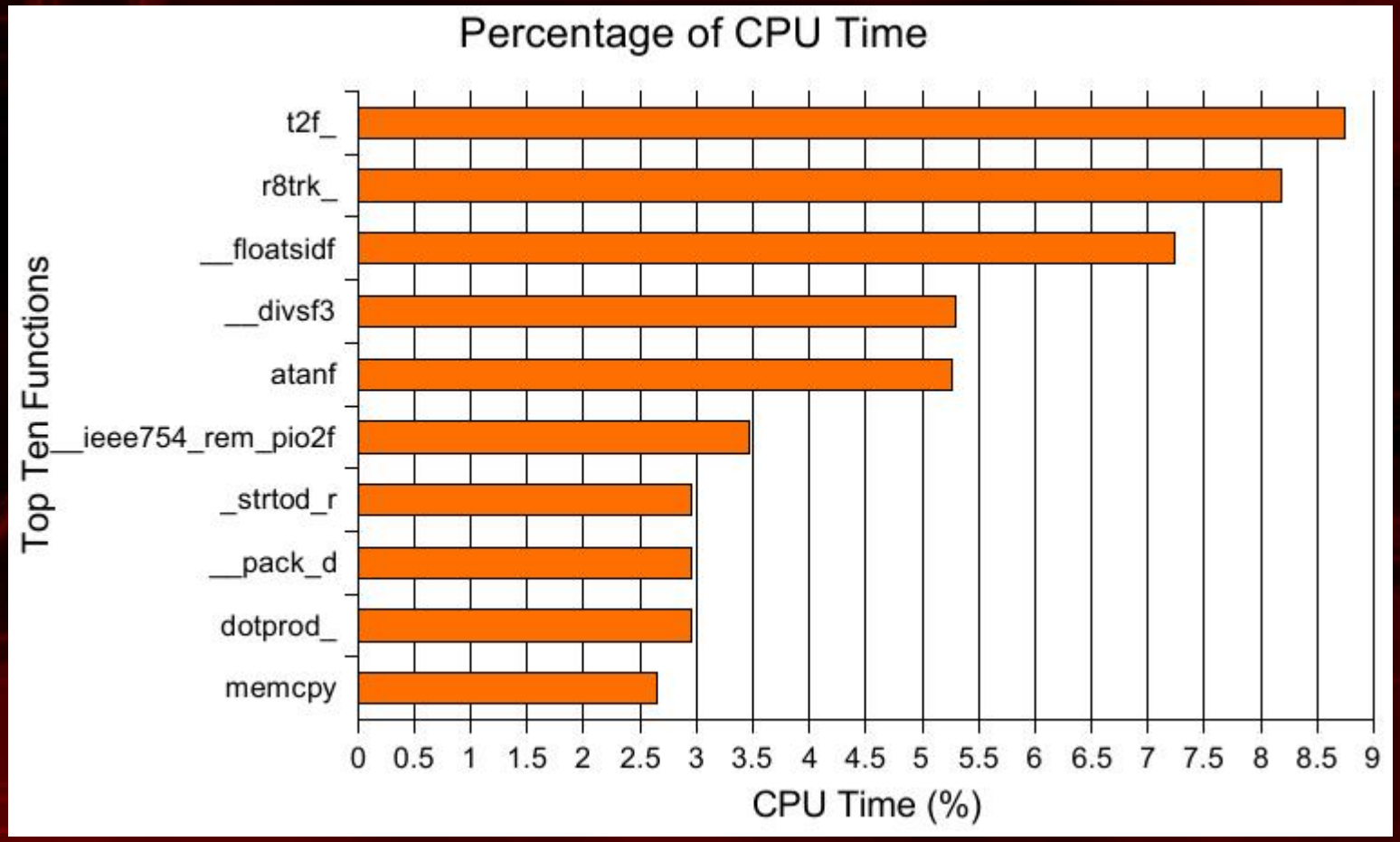

Figure 12: Profiling results for system with dot-prod and APU-FPU co-processor, SP math functions, and DP *Perflib*

Results
All System Builds

CPU (MHz)	FPU (MHz)	Dot-prod (MHz)	SP Math	Perflib	Re-sampling (sec)	Spectrum (sec)	Total (sec)	Speedup
200	-	-	n	-	1197.7938	117.4963	1315.2901	1.00x
200	-	-	y	-	780.9376	109.9881	890.9257	1.48x
200	-	-	y	standard	244.1836	44.4243	288.6079	4.56x
200	100	-	y	DP-only	151.1817	14.3672	165.5489	7.95x
200	-	200	y	standard	211.4479	44.4456	255.8935	5.14x
200	100	200	y	DP-only	149.3297	14.3675	163.6972	8.03x
266	133	-	y	DP-only	114.2044	12.5779	126.7823	10.37x

Table 8: Execution times for all V4FX FTIR system builds

Results
Comparing to Previous Work

	V2P (NASA JPL)		ML410 (Xilinx)		
PPC405 Freq.	300 MHz		266.67 MHz		
Interferograms	104		scaled to 104		
Detectors	2		scaled to 2		
SW Optimizations	IBM Perflib (standard)		SP Math Functions IBM Perflib (DP)		
HW Optimizations	None		APU-FPU (133.33 MHz)		
Software Component	Time (min)	Cycles ($\times 10^{12}$)	Time (min)	Cycles ($\times 10^{12}$)	Speedup
Re-sampling	780	14.040	396	6.336	1.97x
Spectrum (Phase Correction, FFT)	142+90	4.176	44	0.704	5.27x
TOTAL	1012	18.216	440	7.040	2.30x
Efficiency Score	4.11		10.65		-

Table 9: Best performing hybrid-FPGA system compared to previous work at NASA JPL (V2P)

Results
Comparing to Previous Work

	RAD750 (BAE)	ML410 (Xilinx)	
PPC405 Freq.	133.33 MHz	266.67 MHz	
Interferograms	104	scaled to 104	
Detectors	2	scaled to 2	
SW Optimizations	None	SP Math Functions IBM Perflib (DP)	
HW Optimizations	Native FPU	APU-FPU (133.33 MHz)	
Software Component	Time (min)	Time (min)	Speedup
Re-sampling	69	396	0.17x
Spectrum (Phase Correction, FFT)	42+15	44	1.30x
TOTAL	126	440	0.29x

Table 10: Best performing hybrid-FPGA system compared to previous work at NASA JPL (RAD750)

Conclusions

- Results showed significant speedup through SW and HW optimizations of FTIR spectrometry algorithm (10.37x)
- Fastest performance to-date on hybrid-FPGAs (2.30x faster than previous work)
- Still lags behind RAD750 SBC, although margin is smaller
- Developed a dot-product co-processor
  - First implementation of co-processor co-existing with the FPU
  - Not very helpful in FTIR system due to poor data locality

Future Work

- Rewriting entire software in C rather than porting from FORTRAN
- Optimize pattern of data access, improve locality
- Use a different compiler, targeted specifically for PPC405 processor
- Utilize second hard CPU core, with a dedicated co-processors and memory

Acknowledgement

- Charles Norton (JPL)
- Abdullah Aljabri (JPL)

This work was done at RIT as part of a Master's Thesis, which came out of research done at JPL (2006).

The complete Thesis is available here:
http://hdl.handle.net/1850/4805