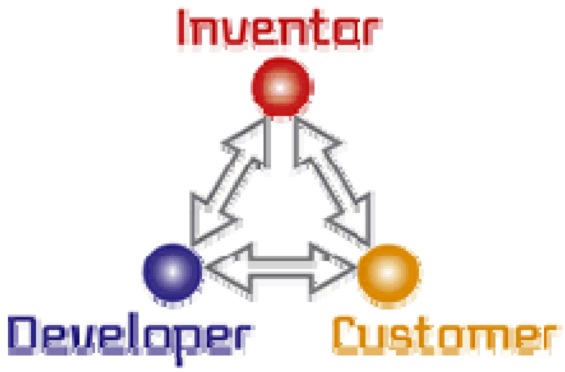
Innovation Triangle Initiative

Multi-threaded processor for space applications - (type B)



Techne Consulting Ltd. Airbus Space & Defence SAS, daiteq s.r.o.

Techne

Innovation Triangle Initiative

Presentatiion Structure

esa

- Overview of the presentation:
  - Why multi-threaded technology and in particular why Microthreading?
  - History of Microthreading
  - Benefits of the technology
  - Implementation results
  - The next steps

Innovation Triangle Initiative

Microthreading

- The aim in this project is to provide flexibility in on-board processing
  - On-board processing typically comprises a mix of programmable cores and dedicated logic
  - Dedicated logic provides performance but at the cost of flexibility and high development costs
- Multi-core processors give performance
  - But still have cost and flexibility issues i.e. in capturing concurrency (rewriting code) and scheduling it to the platform
  - Microthreading captures concurrency just once in the ISA
  - Performance depends on *run-time* allocation of resources, i.e. number of cores / threads without any further development

Techne

daiteq

AIRBUS DEFENCE & SPACE

Innovation Triangle Initiative

Origins of Microthreading

esa

- The original concept of Microthreading dates back to the 1996 A Bolychevsky, C R Jesshope and V B Muchnick, (1996) Dynamic scheduling in RISC architectures, *IEE Trans. E, Computers and Digital Techniques* ,**143**, pp309-317.
  - It was proposed to advance microprocessor design
- Over the last decade two projects saw significant progress of this concept
  - NWO *Microgrids* – funded the development of a cycle-accurate multi-core simulator
  - EU *AppleCORE* funded the development of a basic infrastructure for further development of the concept
    - languages, compilers, multi-core simulators, OS components and a single core FPGA prototype

Techne daiteq AIRBUS DEFENCE & SPACE

Innovation Triangle Initiative

Microthreading Concepts

esa

- Microthreading extends a processor’s ISA to include instructions to capture and manage concurrency in a *hierarchical* manner
- That concurrency is based on families of identical threads, e.g.
  – A single thread family could be a task, a function an interrupt handler, etc. etc.
  – A multiple thread family can represent the concurrent execution of a loop body, with its index range captured and managed in hardware

Techne daiteq AIRBUS DEFENCE & SPACE

Innovation Triangle Initiative

Benefits of Microthreading

esa

- In terms of processor design
  - Microthreaded concurrency can give a high level of latency tolerance on single cores
    - Leading to high pipeline efficiencies and increased energy efficiency e.g. values of 75-95% utilisation are typical
  - Microthreaded concurrency can be distributed over an arbitrary number of cores (within constraints)
    - A scalable multi-core leads to tuneable performance based only on the *run-time* allocation of resources (i.e. cores and threads/core)

Techne

daiteq

AIRBUS DEFENCE & SPACE

Innovation Triangle Initiative

Innovation content

- We believe that this technology can impact on-board processing in a number of ways
  - High pipeline efficiency means energy-efficient computing
  - Well designed multi-core processors can replace expensive and inflexible custom logic with scalable and flexible solutions
  - Unlike standard multi-core processors, systems and applications software will not need costly redesign with each new generation of product
  - It may be possible to integrate system control and payload processing on a common platform

Techne

Innovation Triangle Initiative

Project goals

esa

- The first task was to modify the ISA to optimise resource usage and clock frequency by offloading operations to software that would not materially impact performance
- This revised ISA was then implemented in FPGA
  - The base architecture LEON2-FT core
  - Innovation was separated between LEON2 specific solutions and generic solutions that would be portable to other ISA bases
- This work also required significant updating of the SL toolset, developed in AppleCORE
  - i.e. compilers, simulators and OS components

Techne daiteq AIRBUS DEFENCE & SPACE

Innovation Triangle Initiative

esa

Project goals

- A key goal for this project was to evaluate microthreading based on real hardware using a real payload application
- In the second task therefore AirbusDS recoded the Euclide benchmark algorithm into the core Microthreading language SL
  - This was evaluated for difficulty e.g. man hours
  - The resulting code was run and evaluated on the improved Microthreaded prototype

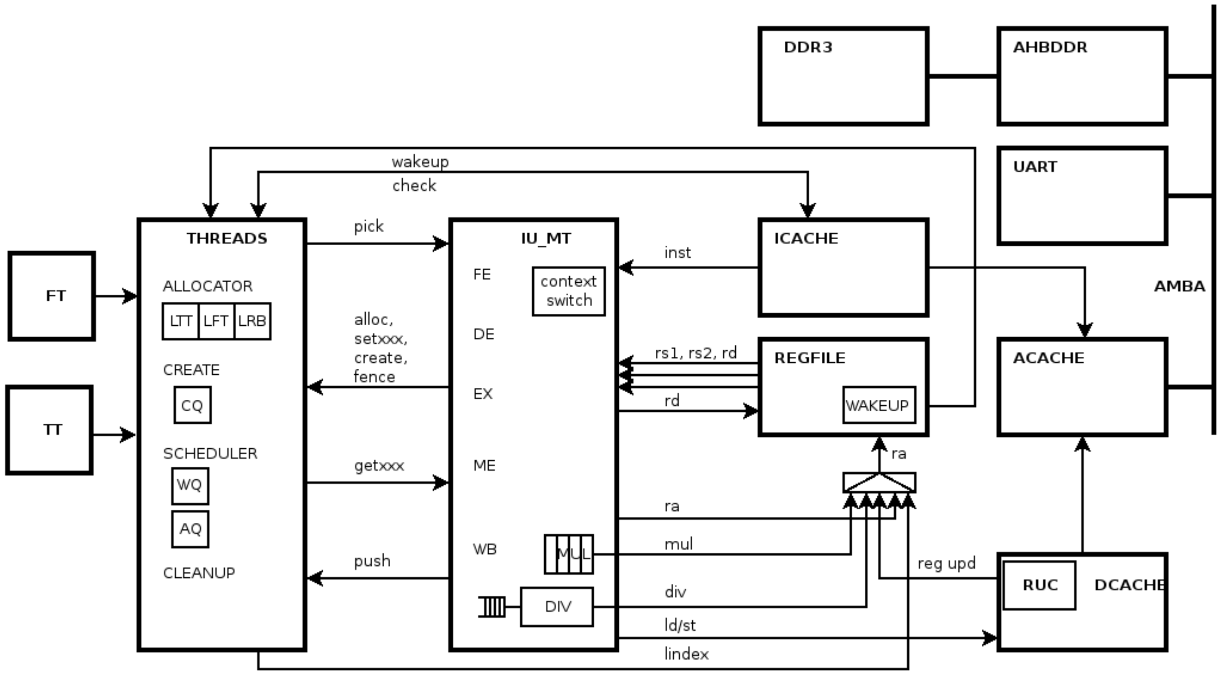
Techne daiteq AIRBUS DEFENCE & SPACE

ISA redesign

- Major changes were made to the ISA were:
  - Remove dependent families used in parallel reductions and offer alternative solution
    - reduced concurrency management logic and simplified register addressing
  - Restrict logical register partitioning (globals/locals)
    - Number of globals was arbitrary now restricted to 0/4/8/16 which optimised further speed of register file addressing
  - Move resource allocation from hardware to software
    - greatly simplifies concurrency management hardware
  - Introduce blocking in index ranges
    - increases hardware complexity marginally but we expect to be able to eliminate known bottlenecks by software tuning

Innovation Triangle Initiative

Prototype



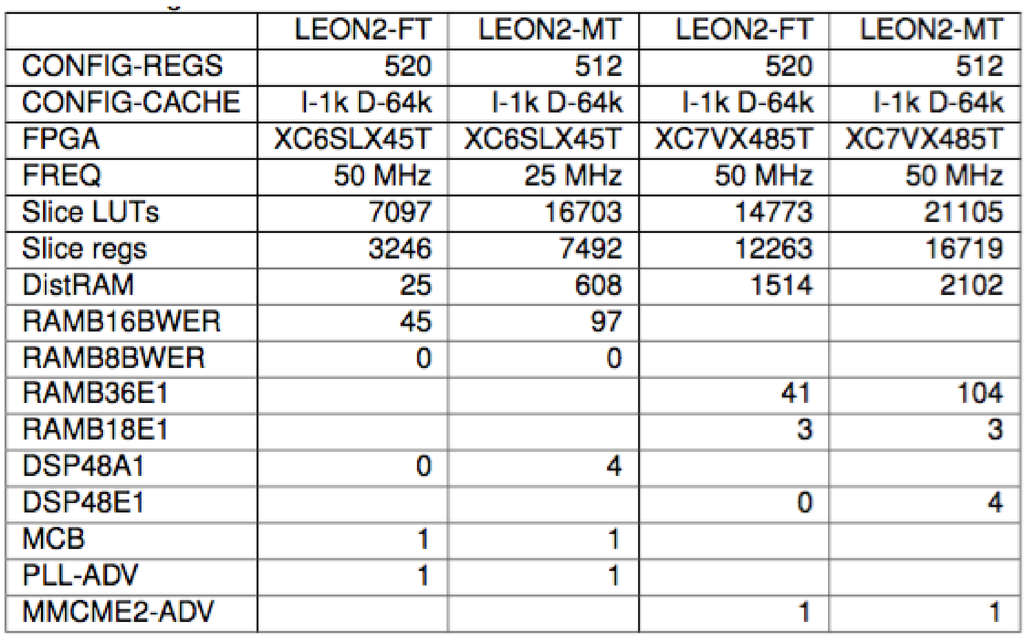
- Overall block diagram of LEON2-MT
- As well as new components to manage the state and schedule threads, changes were also required to the Integer unit, I&Dcache and register file

Techne

daiteq

AIRBUS DEFENCE & SPACE

Resources used



	LEON2-FT	LEON2-MT	LEON2-FT	LEON2-MT
CONFIG-REGS	520	512	520	512
CONFIG-CACHE	I-1k D-64k	I-1k D-64k	I-1k D-64k	I-1k D-64k
FPGA	XC6SLX45T	XC6SLX45T	XC7VX485T	XC7VX485T
FREQ	50 MHz	25 MHz	50 MHz	50 MHz
Slice LUTs	7097	16703	14773	21105
Slice regs	3246	7492	12263	16719
DistRAM	25	608	1514	2102
RAMB16BWER	45	97		
RAMB8BWER	0	0		
RAMB36E1			41	104
RAMB18E1			3	3
DSP48A1	0	4		
DSP48E1			0	4
MCB	1	1		
PLL-ADV	1	1		
MMCME2-ADV			1	1

- Table shows the resources used for Leon2-MT compared to Leon2-FT (same data store for both chips)
- For the larger board where utilisation is not an issue we achieve the same clock speed and use approximately 2X the resources

Techne

Innovation Triangle Initiative

Results - Software

- The Euclid benchmark comprises ~1300 LOCs spread over 10 computation kernels and an input data generator.
- The programmer assigned the task had no experience of Microthreading, the SL language or the environment used prior to this project.

Techne

daiteq

AIRBUS DEFENCE & SPACE

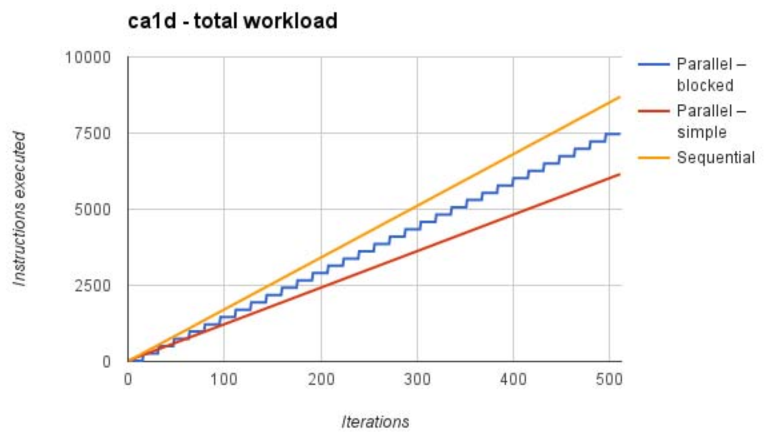
Results - Software

- Total effort in training for the parallelization in SL was 3 days on-site plus approximately one week of self-study
- The complete application was recoded and tested using the simulation environment
- Total effort in recoding the use-case algorithm including testing and debugging was ~3 man-weeks spread over 6 calendar months

Results - Execution

- The results presented here were obtained by running compiled SL code on the prototype core instantiated on a Spartan 6 FPGA board and comprise test kernels, FFT and the Euclide application components
  - For the small workloads (kernels ca1d and rgb2gray-int) there are very few instruction for every store
  - For the larger workloads (rgb2gray-softfp and FFT) the computation to store ratio is larger much in the case of soft fp
- Memory is a potential bottleneck in this design and will need to be redesigned in any future project
  - This is not noticed in LEON2-FT due to the much lower IPC
  - However with expected pipeline efficiencies in LEON2-MT at 75-95, the store bandwidth e.g. one word every 3 cycles for stores and one line in 17cycles for reads has the potential to limit performance

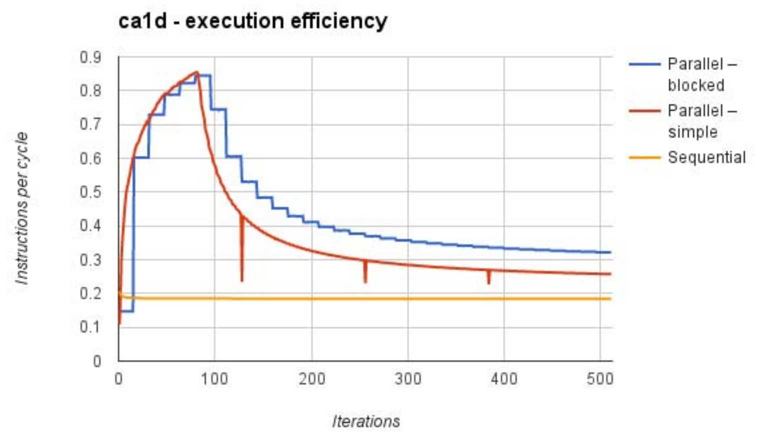
Results – ca1d kernel

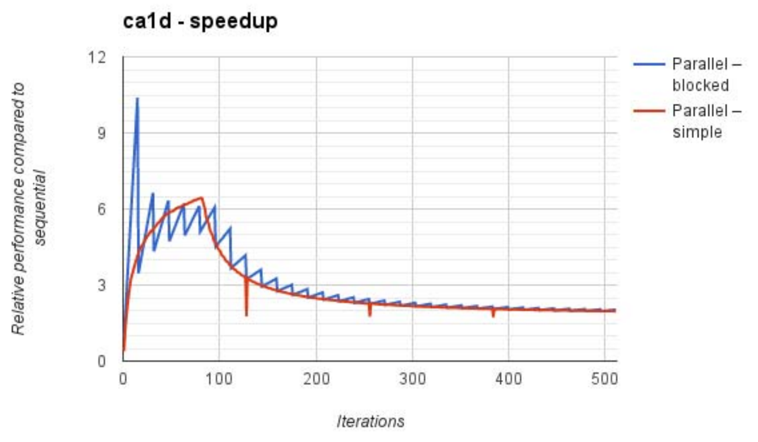


Results show: execution time; execution efficiency (IPC) and speedup vs Sequential

The three cases show: sequential, parallel and parallel blocked, where index blocking is used to constrain locality

This kernel has few instructions in each thread, hence a high proportion of stores which as workload increases limit execution efficiency to memory bus speed (3 cycles per store)



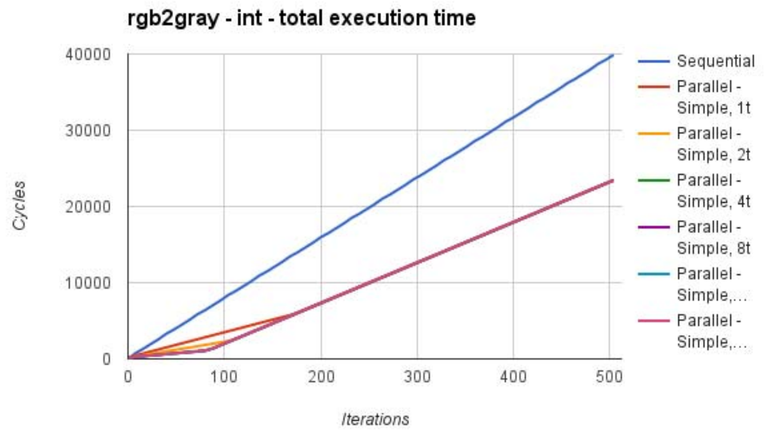


Techne

Innovation Triangle Initiative

Results – rgb2grey kernel

esa

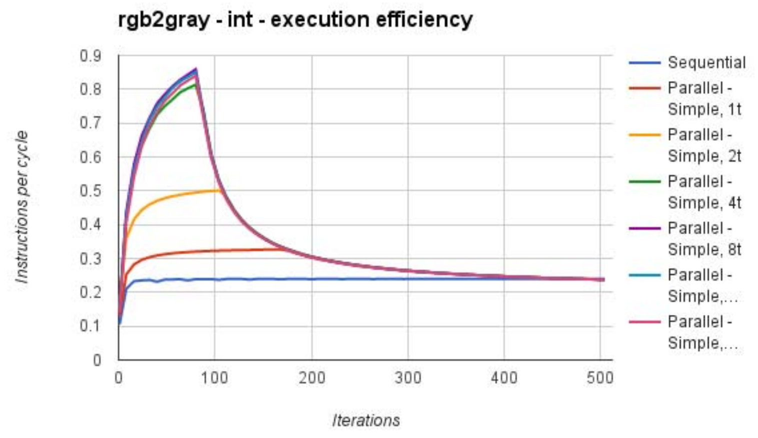


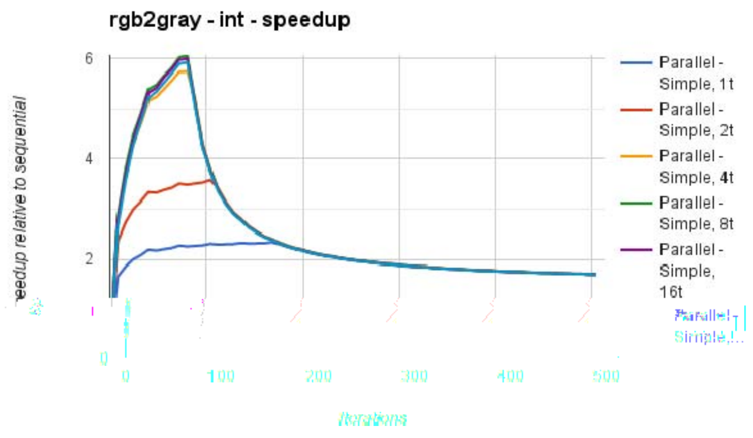
This kernel is also constrained by store pressure

In this kernel we look at the influence of allocation of hardware threads (1, 2, 4, 8, 16, …). There is little difference between 4, 8 and 16 threads

Note a phase change when performance becomes limited by stores at ~90 iterations

For 4+ threads we are getting speedups of ~6X before stores limit performance and ~2 in asymptote





Techne

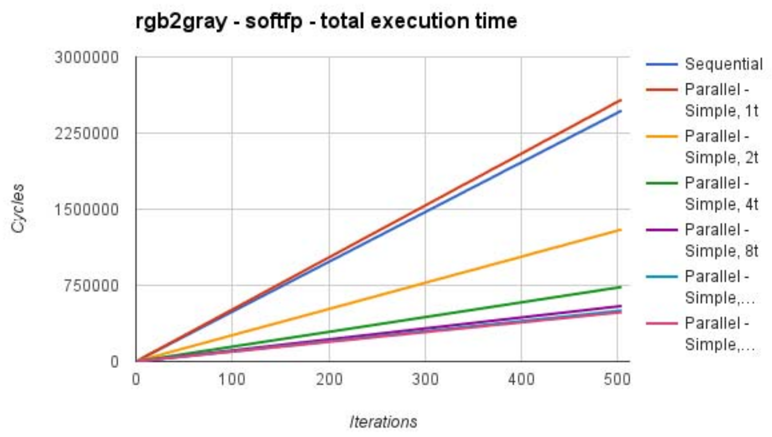
daiteq

AIRBUS DEFENCE & SPACE

Innovation Triangle Initiative

esa

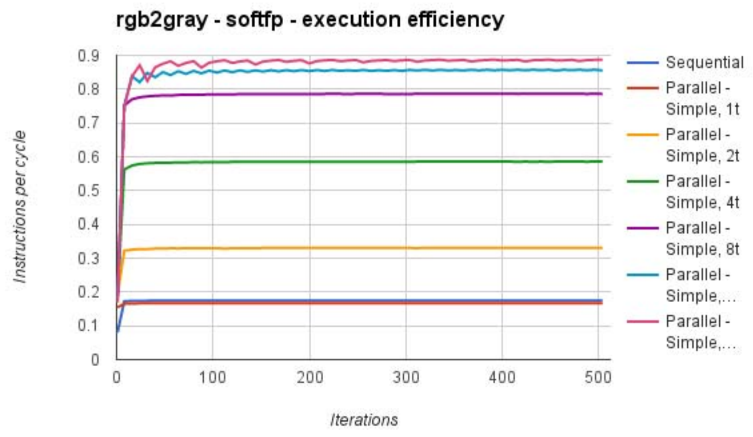
Results – rgb2grey softfp kernel

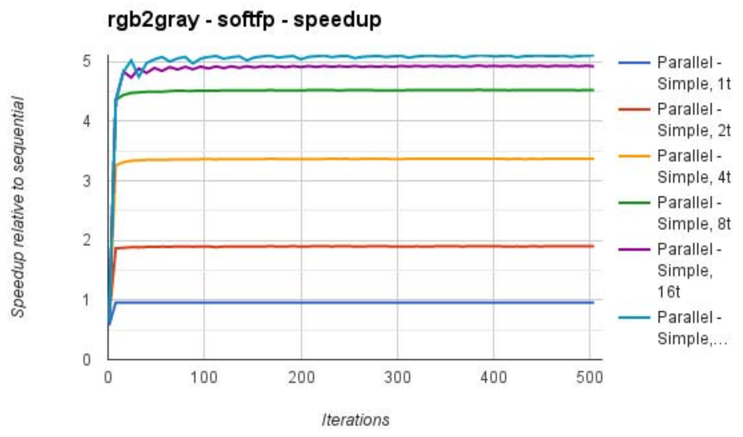


This kernel uses software floating point so f-p operations trap to software and hence increase the ratio of computation instructions to stores.

We see no deterioration of performance as workload increases as stores are more infrequent

Performance increases up to ~16 threads giving speedups of ~5X with pipeline efficiency of ~85%



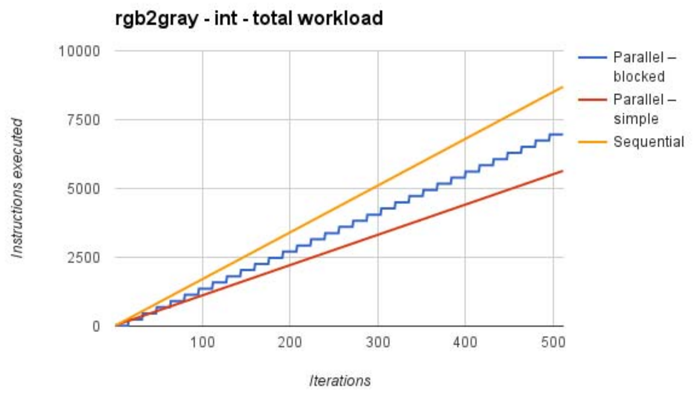


Techne

daiteq

AIRBUS DEFENCE & SPACE

Analysis



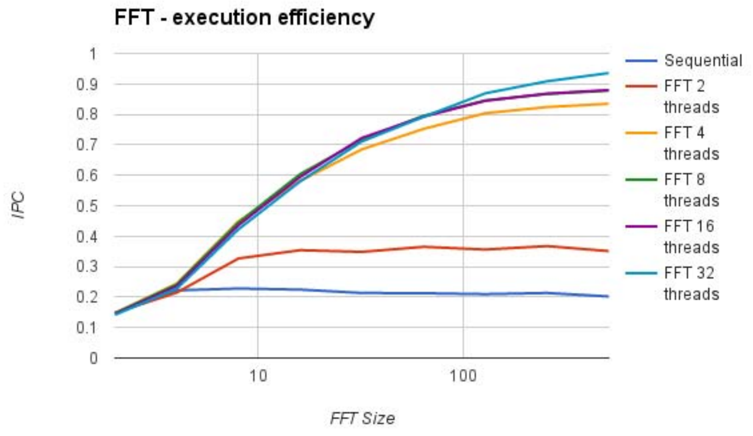
- Speedups for few iterations in the int kernels are enhanced because Microthreading reduces the overall workload (see above)
  - index increment as well as loop bound checks are performed in hardware
- The subsequent drop in execution efficiency we believe is due to the effects of the store buffer bottleneck
- With more computation between stores, e.g. when using software floating point in rgb2grey the store buffer is not a bottleneck
- Here we see speedup increasing with number of threads up to 4.5X

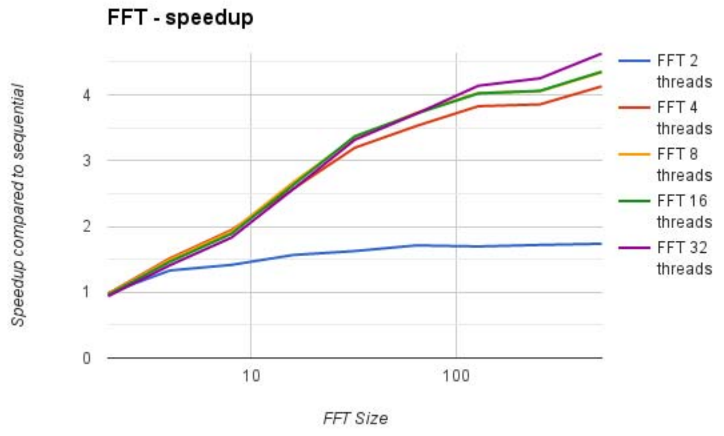
Techne

Innovation Triangle Initiative

Results – FFT

- FFT is a complex algorithm which has poor locality in Dcache in a number of its logn stages (access strides are 1, 2, 4, 8, etc.)
  - we see execution efficiencies of 80-90% for 4 or more threads
  - we see speedups of up to4X for 4 or more hardware thread as the FFT size increases
- N.b because of the log scale an FFT of 32 points, i.e. just 16 software threads falls in the middle of the scale





Techne

daiteq

AIRBUS DEFENCE & SPACE

Innovation Triangle Initiative

Results – Euclide

- Due to a hardware bug it was not possible to run the complete application on the prototype core
- Various components of the application were executed successfully over a range of application and resource parameters
  - However different components failed on different parameters precluding any parallel execution of the complete application

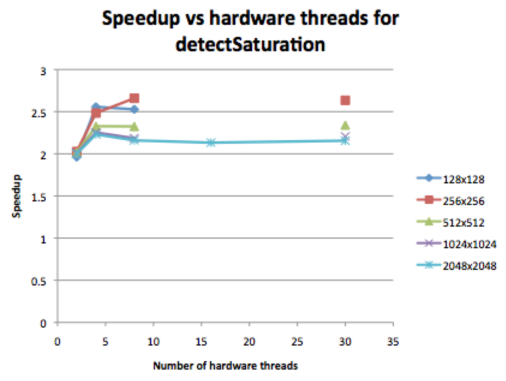
Techne

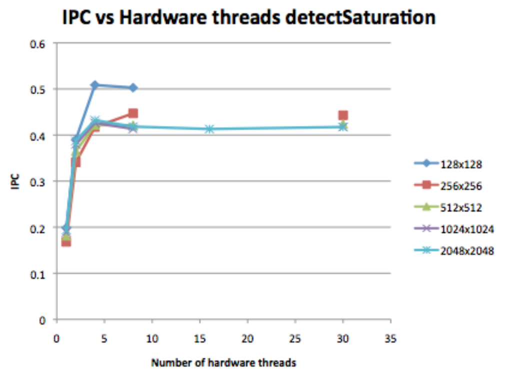
daiteq

AIRBUS DEFENCE & SPACE

Results – Euclide

- The fragmented results are presented in the report in detail - here we focus on two components
- *detectSaturation* shown right, which exploits the new feature parallel supporting reductions
  - It can be seen that even with relatively few threads we see speedups of 2 to 2.5 and efficiencies approaching 50%
- *detectCosmicRays* which is the main component of Euclide shown in the table





Hardware threads	Cycles	Instructions	Frame size	IPC	Speedup
1	2145519	340308	128x128	0.16	
2	1166021	338446	128x128	0.29	1.83
1	171865782	21252828	512x512	0.12	
4	26966984	18291473	512x512	0.68	5.49

Techne

Innovation Triangle Initiative

Further developments

esa

- Our results demonstrate the benefits of this technology
  - high pipeline efficency and speedup over the conventional core are due to being able to tolerating large latency operations
  - flexibility in deployment - run code using 1-16 hardware threads on one core without any change to code or recompilation
  - an implementation with a similar clock frequency compared to the Leon2 using ~ 2X resources (depending on configuration)
  - Compared to the Apple-CORE prototype we see 15X in overall design efficiency using the measure - freq*IPC/resources
- However, this is still a single core and to demonstrate the full benefits of tuneable performance up to custom logic speed we need to implement a multi-core prototype

Techne

daiteq

AIRBUS DEFENCE & SPACE

Innovation Triangle Initiative

Further developments

- The tasks required in designing a multi-core are:
  - specify the architecture and communication layers, i.e. distributed register file, cache hierarchy and cache coherence
  - design the multi-core architecture
  - verify the multi-core architecture
  - evaluate the prototype and characterise performance against resources used
  - investigate the use of mixed criticality applications on the same platform
- Cost around 250KEUR

Techne daiteq AIRBUS DEFENCE & SPACE

Innovation Triangle Initiative

Further developments

esa

- Alternatively a more aggressive project could develop an ASIC multi-core implementation
- This would require the same design steps but in addition it would be prudent to also put more into software development
- Examples are to cross compile to SL from other more standard languages and/or intermediate languages, e.g. OpenCL or Single Assignment C (SaC)
- Project cost ≥1MEUR

Techne daiteq AIRBUS DEFENCE & SPACE

Innovation Triangle Initiative

Further developments

esa

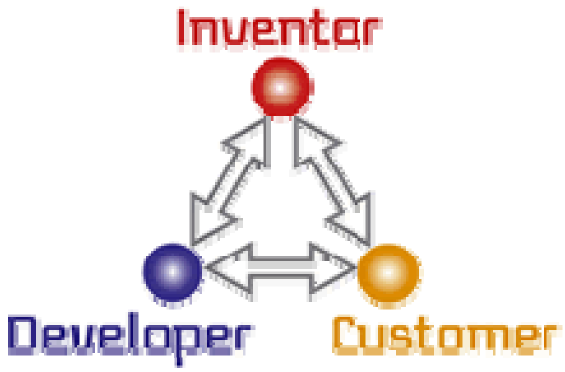
- This is foundation technology and if adopted has wide ranging use in space applications
- Anticipated spin offs include the use of the same technology in other compute intensive/energy restricted fields
  - E.g. data-centres … but to compete with current processor technology either innovative solutions need to be adopted or a large investment made in a custom core implelemtation

Techne

daiteq

AIRBUS DEFENCE & SPACE

Unused Slides



Innovation Triangle Initiative

Benefits of Microthreading

esa

- In terms of System design
  - Code is redeveloped once only, compiled and then performance is tuned by parameterising the resources used to execute it
  - Preceding Type A project investigated two levels of priority threads by simulation and it was found:
    - a high priority thread gives extremely low latency and jitter even running heavy background computations
      - e.g. 35-180μsec from interrupt for a 30μsec nominal task at 10MHz
    - running high priority threads periodically did not significantly impact the background computational tasks

Techne

daiteq

AIRBUS DEFENCE & SPACE

Innovation Triangle Initiative

Project goals

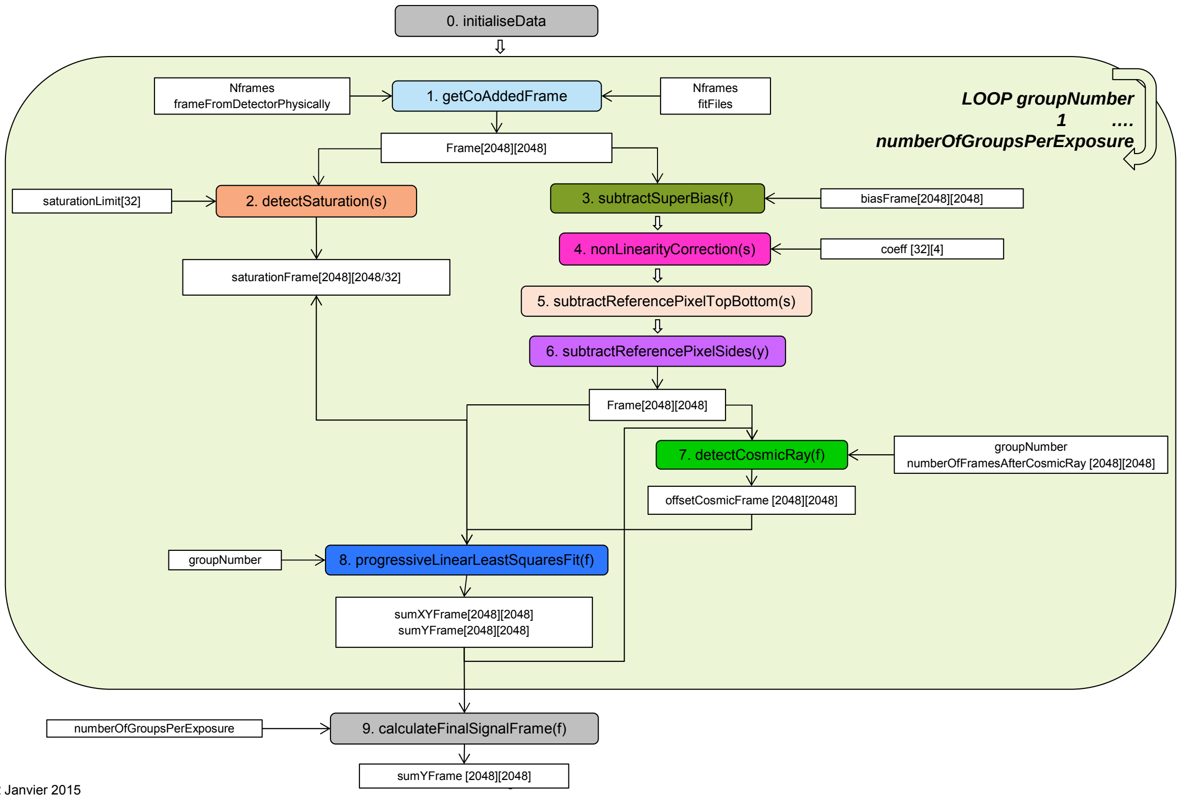
esa

- The AppleCORE project produced an FPGA prototype of a microthreaded core: *UTLeon3* based on the Leon3 core
  - the overall relative design efficiency defined by:
    - Pipeline-effiency*clock-speed/gates-used
  - was ~0.14 for FPGA and ~0.18 for ASIC, *hence UTLeon3 was 14-18% as efficient as the Leon3*
- There were known inefficiencies in this design so the first task was to redesign the ISA to achieve a more efficient prototype implementation

Techne daiteq AIRBUS DEFENCE & SPACE

Results - Software

- The computation kernels contain a mix of purely data-parallel operations with equal strides, data-parallel with mixed stride, and reductions.
- The access patterns, while relatively regular, put pressure on the cache memory system and suffer heavy miss rates in sequential execution.


0. initialiseData
Nframes frameFromDetectorPhysically
1. getCoAddedFrame
Nframes fitFiles
LOOP groupNumber 1 .... numberOfGroupsPerExposure
Frame[2048][2048]
saturationLimit[32]
2. detectSaturation(s)
3. subtractSuperBias(f)
biasFrame[2048][2048]
4. nonLinearityCorrection(s)
coeff [32][4]
saturationFrame[2048][2048/32]
5. subtractReferencePixelTopBottom(s)
6. subtractReferencePixelSides(y)
Frame[2048][2048]
7. detectCosmicRay(f)
groupNumber numberOfFramesAfterCosmicRay [2048][2048]
offsetCosmicFrame [2048][2048]
groupNumber
8. progressiveLinearLeastSquaresFit(f)
sumXYFrame[2048][2048] sumYFrame[2048][2048]
numberOfGroupsPerExposure
9. calculateFinalSignalFrame(f)
sumYFrame [2048][2048]

Techne