Crypto On the Cell

Neil Costigan

School of Computing,
Dublin City University.

neil.costigan@computing.dcu.ie +353.1.700.6916

PhD student / 2nd year of research.

Supervisor : - Dr Michael Scott.

IRCSET funded.

Talk Overview

- Cell background
- Our work
- Other work
- Folding@home
- Other processors
- Our next steps

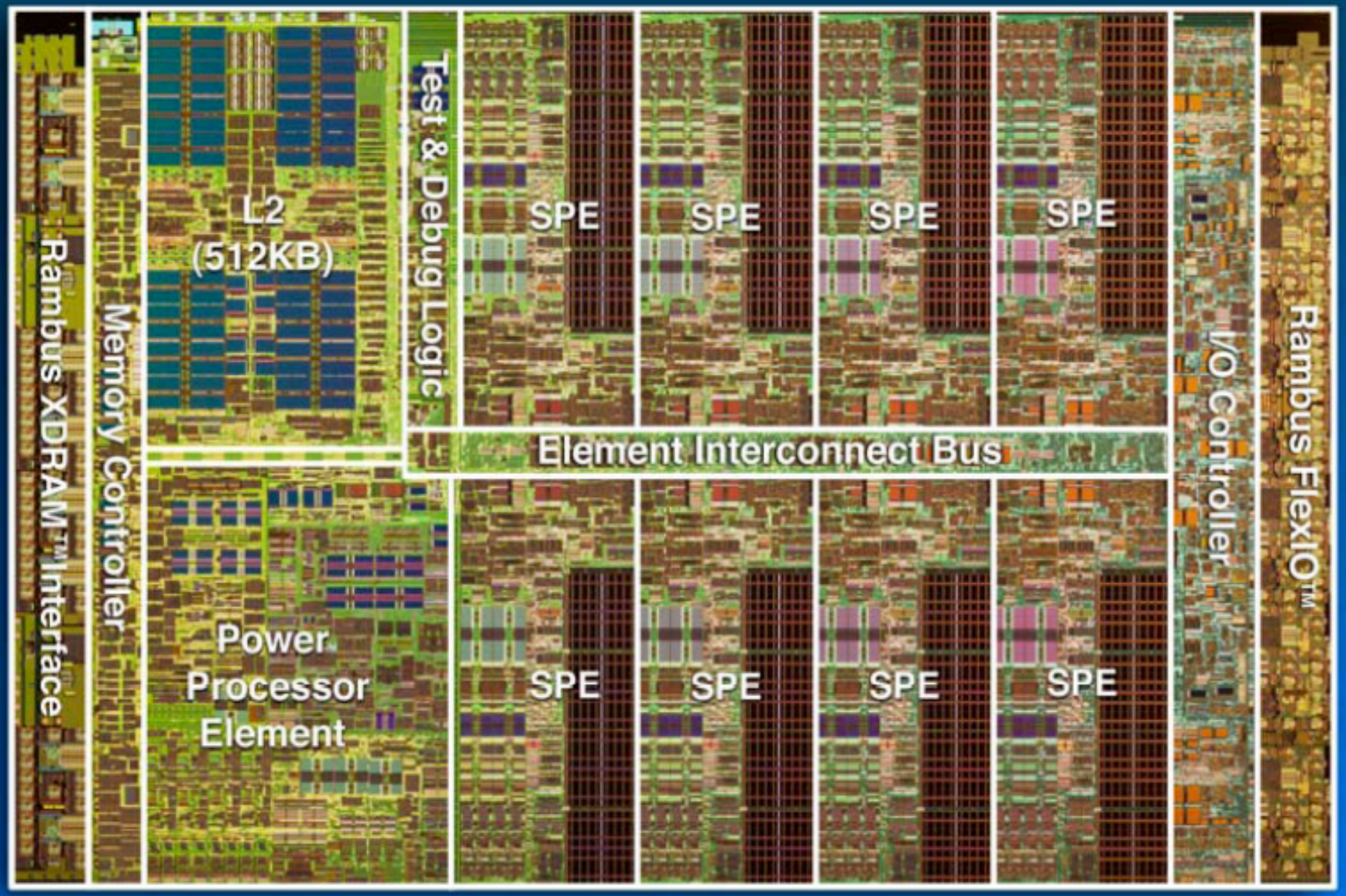
Playstation 3

- Background
  - Sony, IBM, Toshiba
- Cell Broadband Engine
- Multipurpose
- Linux
- Development environment

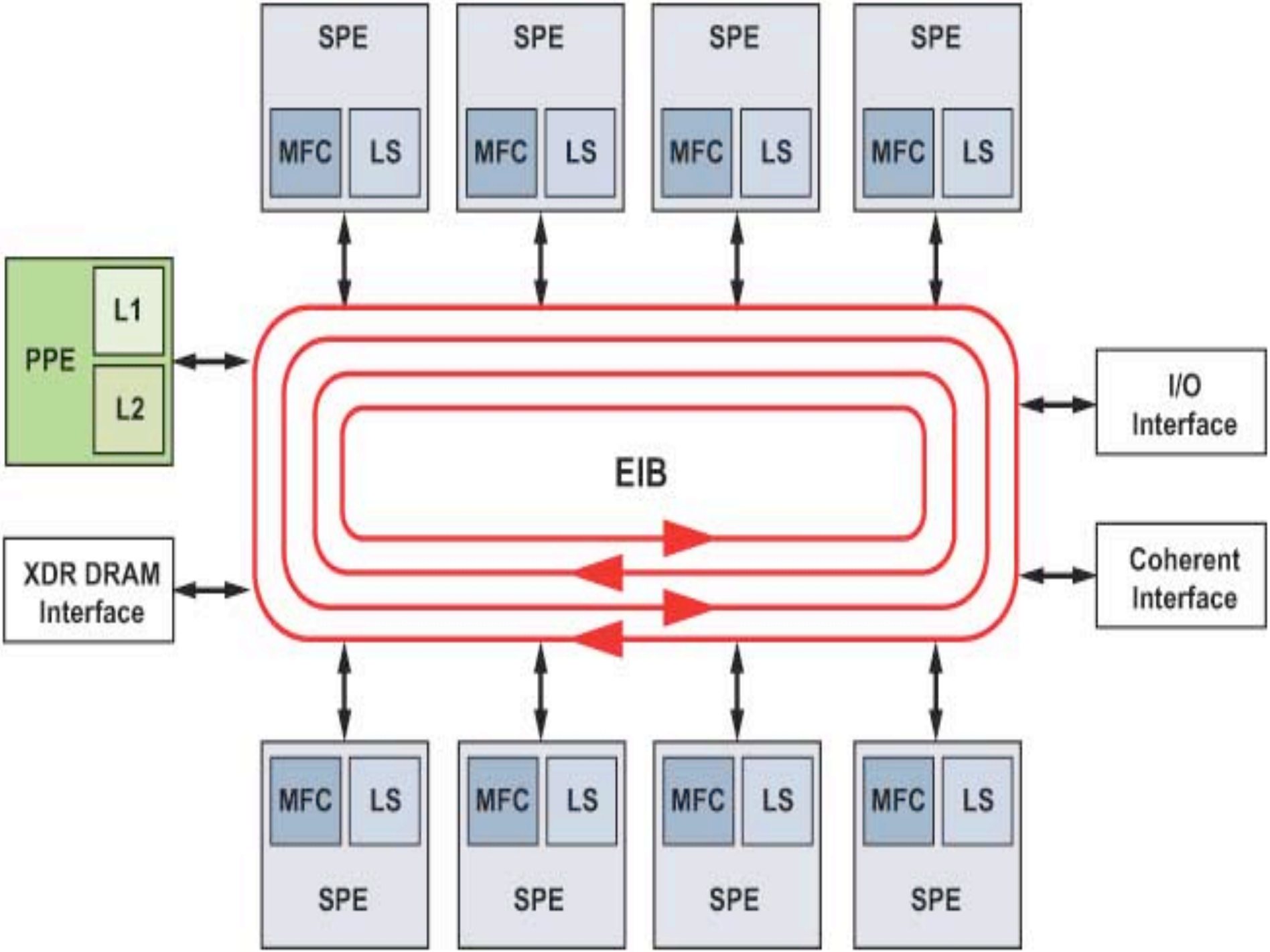
Cell Broadband Engine

- A 9-way multiprocessor
- One main 64-bit PPE processor
  - Power Processor Element, 2 hardware threads
  - Good at control tasks, task switching, OS-level code
- 8 SPE processors
  - Synergistic Processor Element
  - Good at compute-intensive tasks

Cell Broadband Engine Processor



IBM


SPE
MFC
LS
SPE
MFC
LS
SPE
MFC
LS
SPE
MFC
LS
PPE
L1
L2
EIB
I/O
Interface
XDR DRAM
Interface
Coherent
Interface
MFC
LS
SPE
MFC
LS
SPE
MFC
LS
SPE
MFC
LS
SPE

Playstation Vs Cell Blade

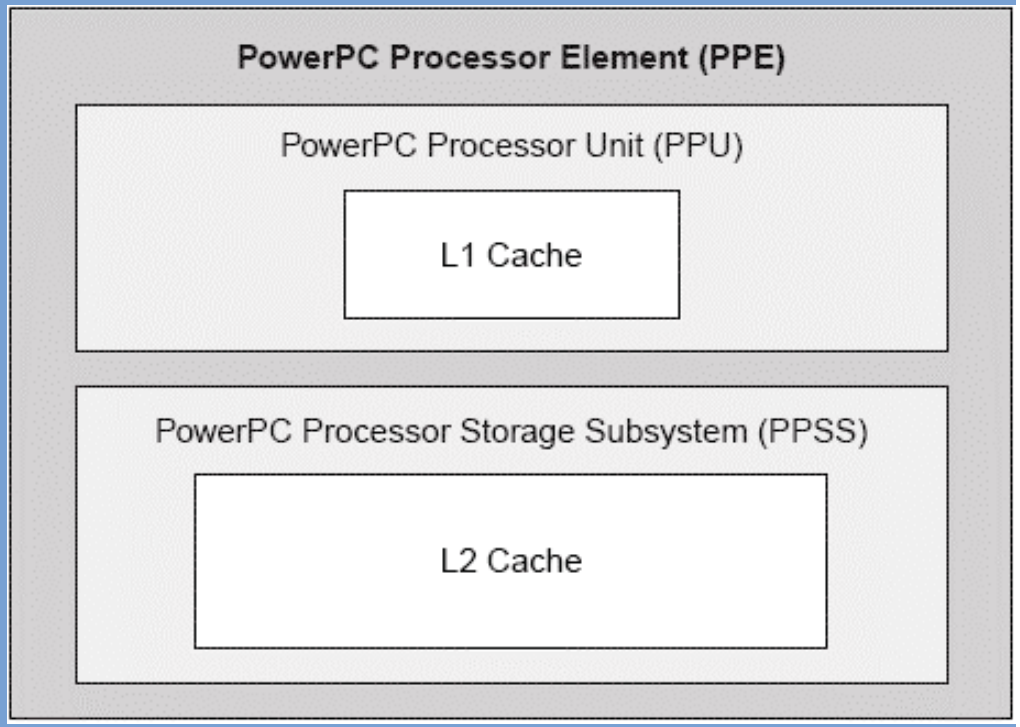
- 6 vs 8 SPE
- Cell blade can be configured for dual Cell
- All 16 SPEs visible to each PPE

Whats in a name ?

SPU / SPE ?
PPU / PPE ?

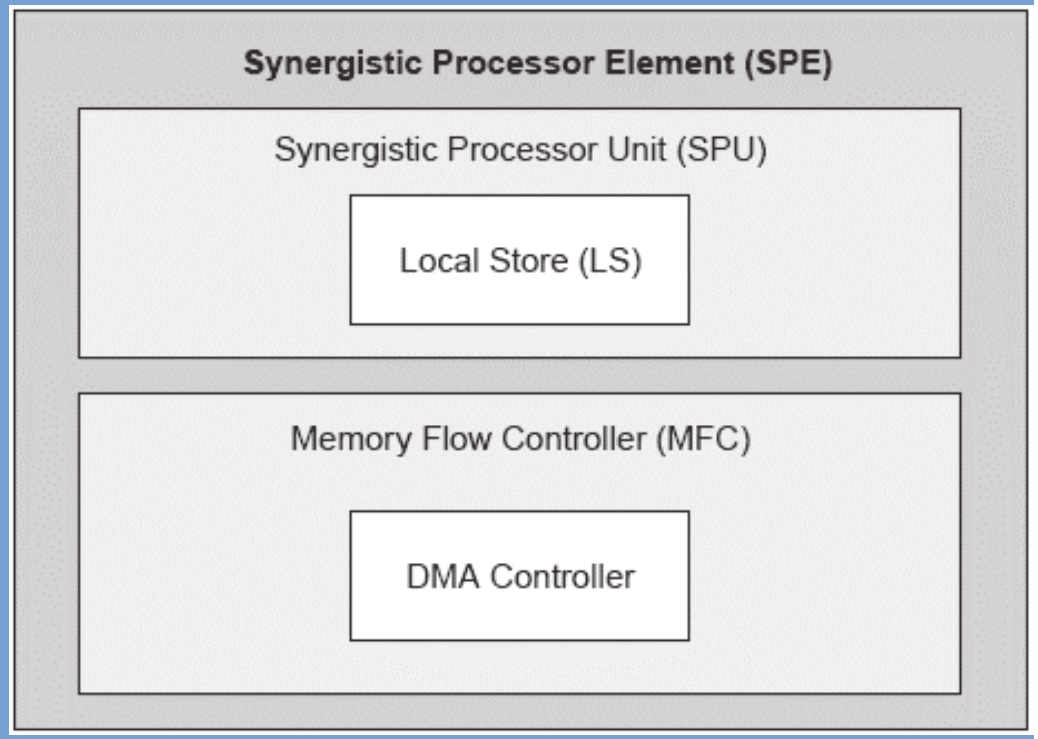
Power Processor Element

- 64-bit PowerPC Architecture
- RISC core
- Tradition virtual memory subsystem
- Supports Vector/SIMD instruction set
- Runs OS, manages system resources etc



Synergistic Processor Element

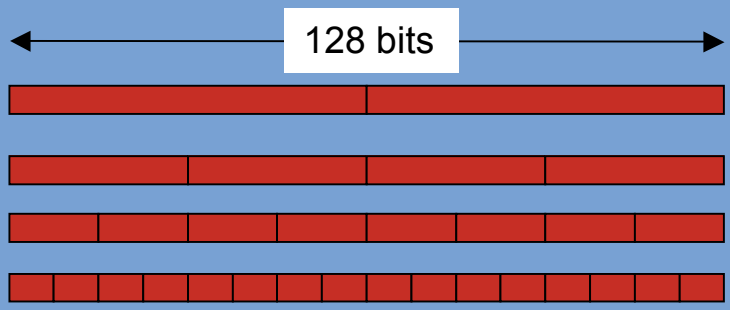
- RISC core
- 256kb local store
- 128-bit, 128 entry register file
- Vector/SIMD
- MFC controls DMAs to/from Local Store over EIB

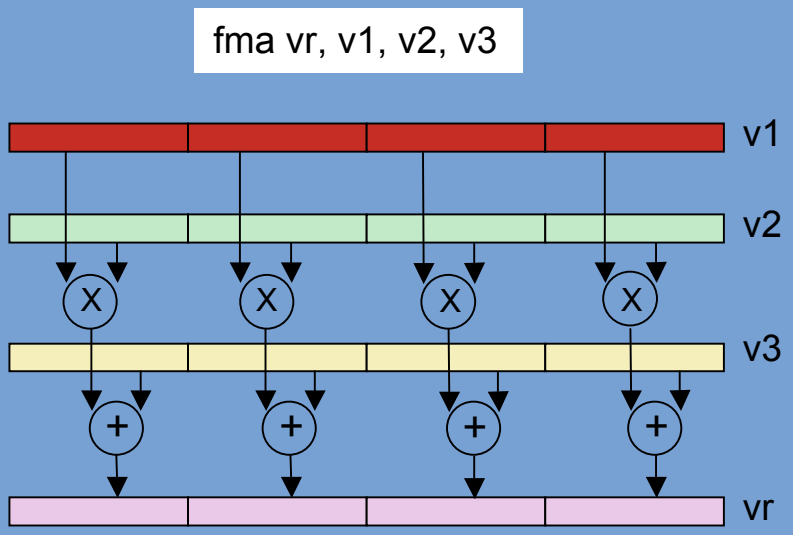


So what ?

- 128 * 128-bit registers is the key.
- Up to 4 * 32-bit integer operations in ***one*** clock cycle.
- 2 instruction pipelines
- Overlapped DMA

- note
  - it is just a 16x16bit multiplier





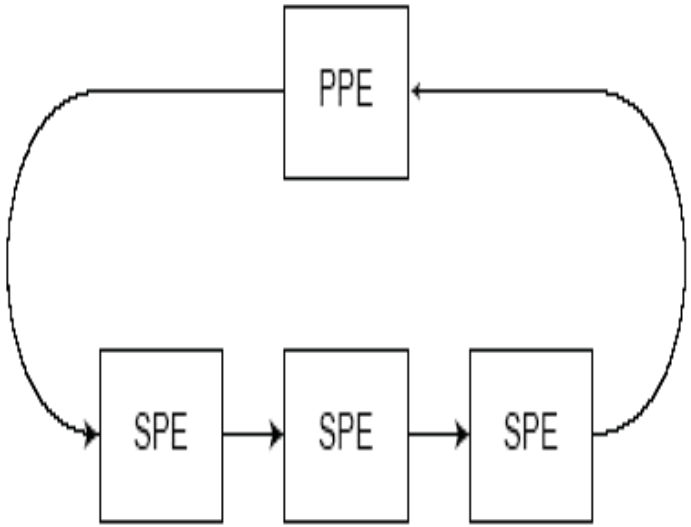
Hardware Security

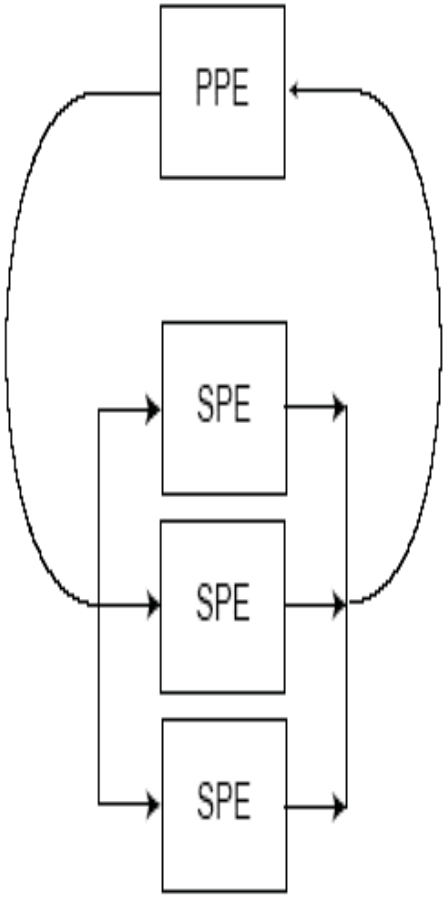
- Secure processing vault (SPV):
- Runtime secure boot
- Hardware root of secrecy
- Hardware random number generator (RNG):

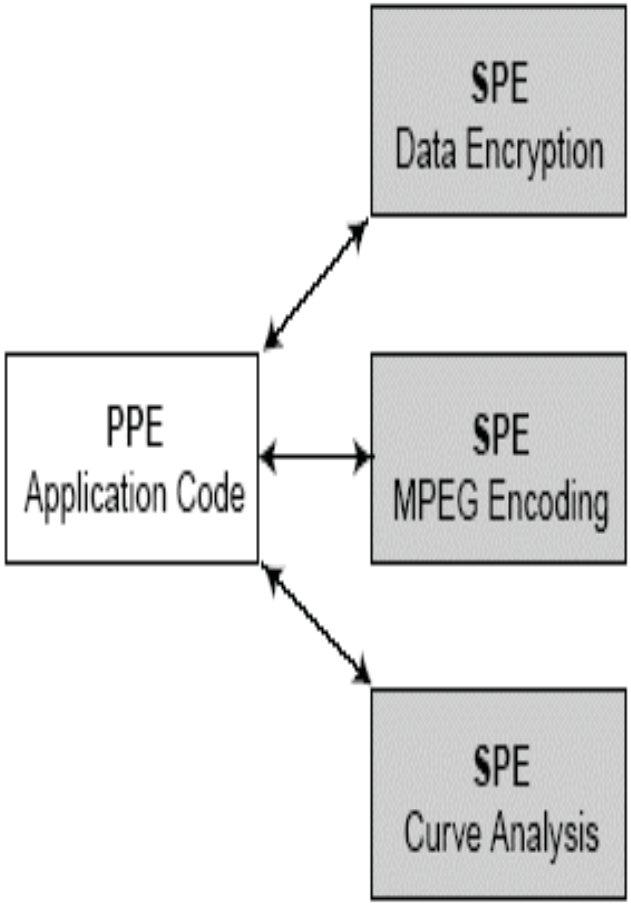
Roadmap

- Fast double precision float.
  - Available in the 2.1 Simulator.
- Recently announced SDK 3.0 and x86 add on boards.
- More local store memory in SPUs
- More SPUs
- Clock speeds > 5Ghz

Usage models







Multistage parallel services

Build process

Code : ASM like ADD

Assembly-like example of a speed up technique when adding a 128-bit value to a 64-bit value where we know there is no need to manage an overflow.

This technique is used in summing partial products inside the big number multiply.

```
vector unsigned int _out_s, _in_a128, _in_a64;
vector unsigned int _sum, _c0, _t0;

_c0 = spu_genc(_in_a128, _in_a64); // generate carry bits
_sum = spu_add(_in_a128, _in_a64); // add
_t0 = spu_slqwbyte(_c0, 4); // shift quadword left 4 bytes
_out_s = spu_add(_sum, _t0); // add in the carry
```

Pipelines

- Compliers not optimal
- Pipelines are not 'equal'
  - Load/store : Odd
  - Shifts etc. : Odd / 4
  - Integer multiply : Even / 7
- Hand code intrinsics.

DMA

- The major programming task is DMA management

- SPE pulls data to LS
- Processes
- Pushes results back to main memory
- Signals completion.

- Overlapping !

What we are doing…

- Looking at the Cell BE for number crunching
- Using the vector processors to improve crypto performance.
- Currently with native vector multi-precision library

OpenSSL project

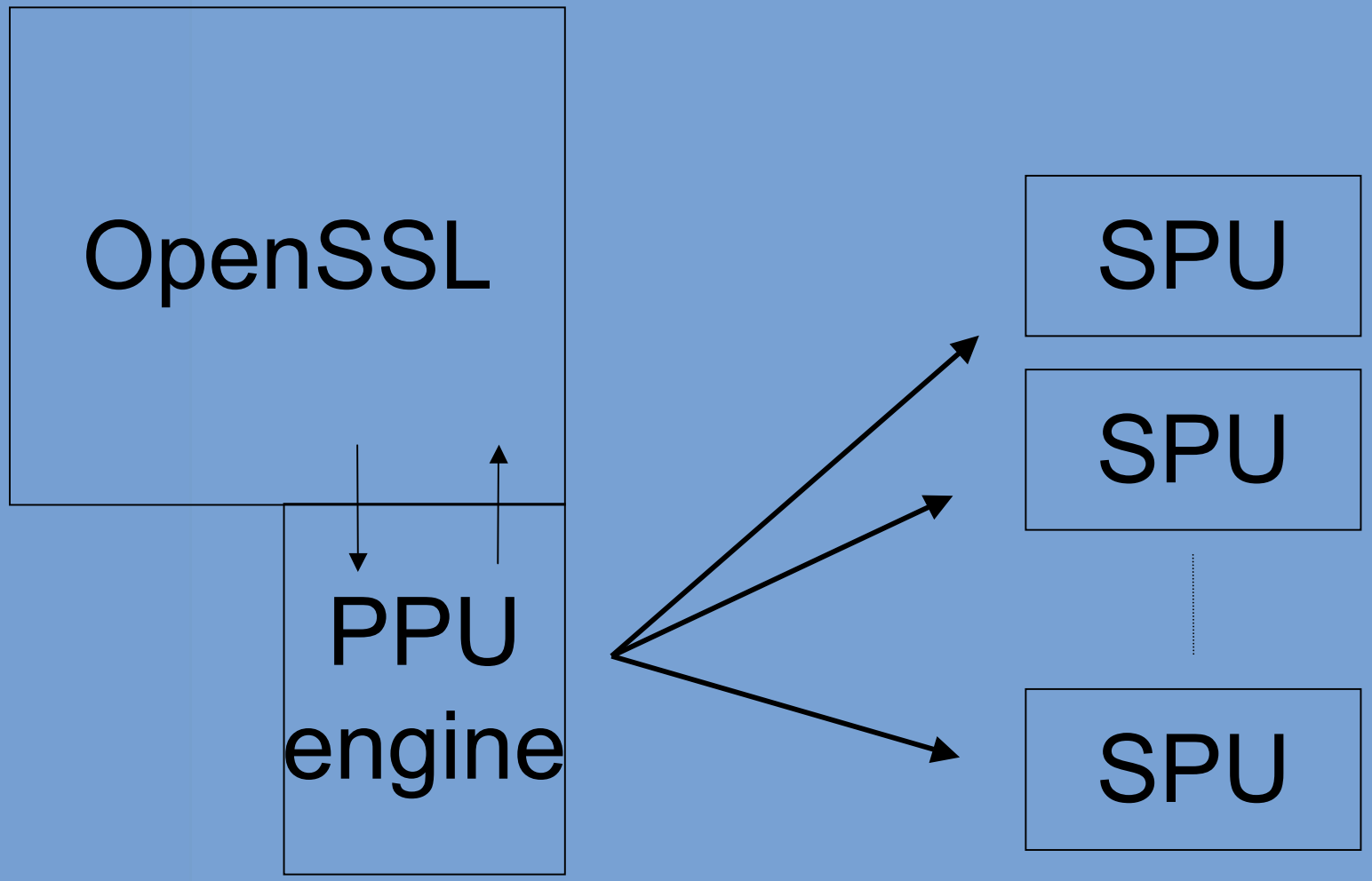
- PKI & crypto toolkit.
- Ships with all Unix variants (Linux, MacOSX etc.)
- Apache, MySQL etc.
- Ability to offload some algorithms via engine interface (like a plug-in)
- We choose RSA with CRT
  - Analysis of SSL points to it being ~95% of session overhead.
  - Interesting as it has two independent mod_exp() that can be run in parallel.

RSA / CRT

```
INPUT: p, q, I0, dmq1, dmp1, iqmp
OUTPUT: r0

r1 <- I0 mod q
m1 <- (r1 ^dmq1) mod q
r1 <- I0 mod p
r0 <- (r1 ^dmp1) mod p
r0 <- r0 − m1
if r0 < 0 then
   r0 <- r0 + p
end if
r1 <- r0 · iqmp
r0 <- r1 mod p
r1 <- r0 · q
r0 <- r1 + m1
```

By counting clock cycles we get **14.87** 4096-bit signs/sec

Architecture 1



RSA : reference / Intel

RSA key length	Linux P4 2.5Ghz sign/sec	MacOSX 2.3Ghz Duo sign/sec
1024-bits	151.1	297.4
2048-bits	26.7	50.4
4096-bits	4.4	7.6

- OpenSSL speed / multi 6 / elapsed time
- Linux : Intel 32-bit P4 2.4Ghz
- MacOSX 10.4 : Intel Core 2 Duo 32-bit 2.3Ghz
- Using native OpenSSL with ASM

RSA : PPU vs. 1 SPU

RSA	PPU		1 SPU	
key length	sign	sign/sec	sign	sign/sec
1024-bits	0.003435s	291.2	0.005655s	176.8
2048-bits	0.017541s	57.0	0.015636s	64.0
4096-bits	0.109793s	9.1	0.070915s	14.1

- OpenSSL speed on PPU vs. 1 SPU
- using IBM-MPM
- 3.2GHz Cell.

RSA : PPU vs. 6 SPU

RSA	PPU		6 SPU	
key length	sign	sign/sec	sign	sign/sec
1024-bits	0.000724s	384.5	0.001906s	524.7
2048-bits	0.002600s	71.7	0.003033s	329.7
4096-bits	0.089455s	11.2	0.011925s	83.9

- OpenSSL speed on PPU vs. 6 SPUs
- using IBM-MPM
- 3.2GHz Cell, 6 parallel processes.

RSA : 2 PPU vs. 16 SPU*

RSA	2 PPU		16 SPU	
key length	sign	sign/sec	sign	sign/sec
1024-bits	0.001270s	787.5	0.001509s	745.1
2048-bits	0.006805s	146.9	0.001664s	601.0
4096-bits	0.053944s	22.8	0.005762s	173.6

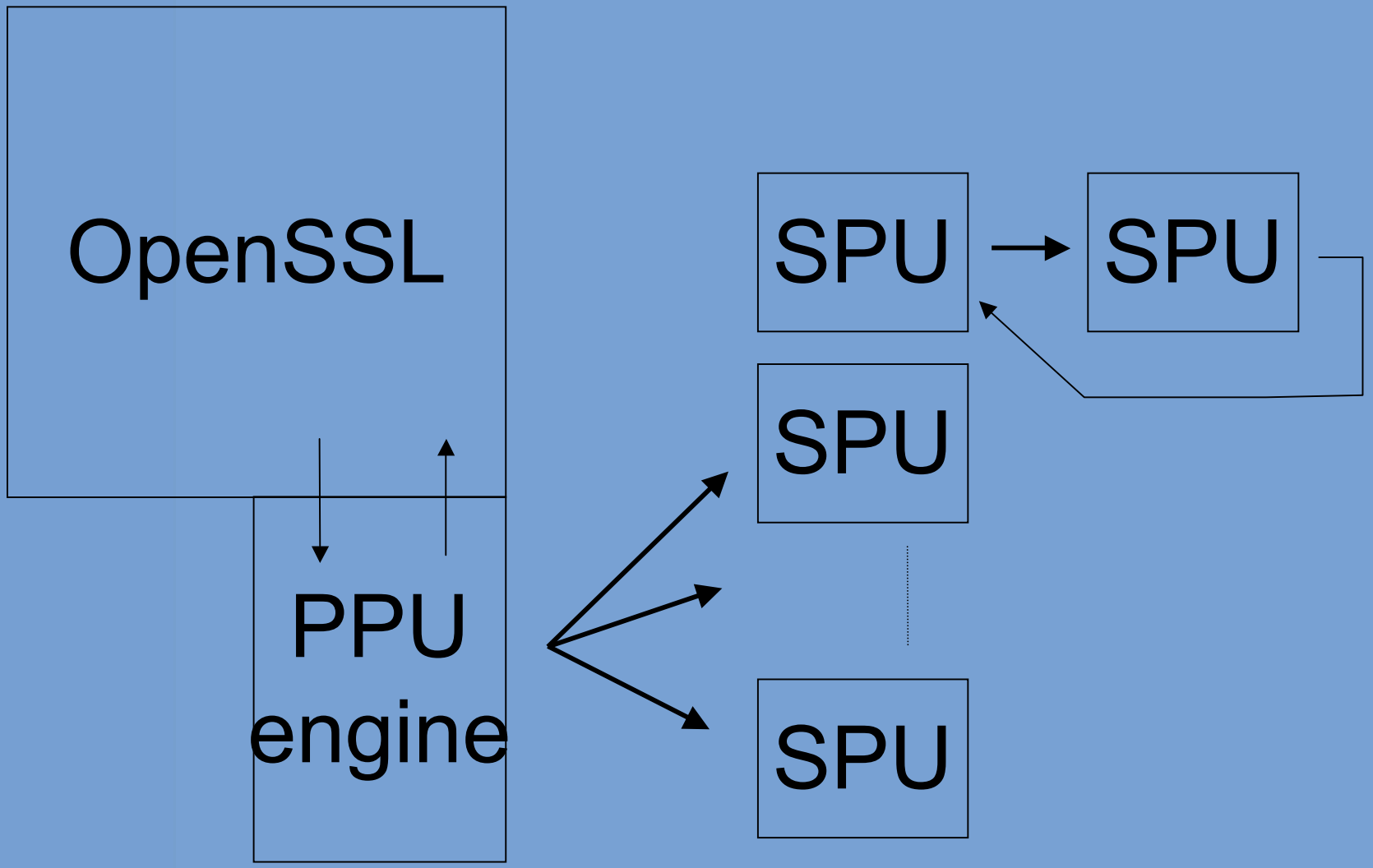
- OpenSSL speed on cell dual PPU with 16 SPUs
- using IBM-MPM
- 3.2GHz Cell, 16 parallel processes.

- * numbers are from one run by IBM.

Architecture 2

```
INPUT: p, q, I0, dmq1, dmp1, iqmp
OUTPUT: r0
        PPU                              SPU 1                    SPU2
r1 <- I0 mod q

THREAD                              m1 <-(r1 ^dmq1)mod q

r1 <- I0 mod p

THREAD                                                r0 <-(r1 ^dmp1) mod p

WAIT

r0 <- r0 − m1
if r0 < 0 then
   r0 <- r0 + p
end if
r1 <- r0 · iqmp
r0 <- r1 mod p
r1 <- r0 · q
r0 <- r1 + m1
```

Architecture 2



RSA : PPU vs. 2 SPU parallel

RSA	PPU		2 SPU	
key length	sign	sign/sec	sign	sign/sec
1024-bits	0.003435s	291.2	0.005208s	192.0
2048-bits	0.017541s	57.0	0.009775s	102.3
4096-bits	0.109793s	9.1	0.037392s	26.7

- OpenSSL speed on PPU vs. 2 SPU parallel mod_exp()
- using IBM-MPM
- 3.2GHz Cell.

Next steps…

- OpenSSL native uses Karatsuba method but IBM Library doesn't. (x3 ?)
- Plan is to port MIRACL to get this
- Look at OpenSSL pre-release for improving ALL algorithms.
- 256bit mod multiply

Closer..

- the fundamental speed up is to muldvd() MADD()
- r = a * b + c / (64bit * 64bit + 64bit = 128bit)

- Remember there is no 128bit C type

- we have a CPI of .7 for about 140 instructions
- current killer is dependency stalls related to shifting in and out of vectors for the unsigned long long type

- exercise now is to unroll big number code (e.g. 1024bit) and propagate through the crypto lib to
  - optimise the carry management
  - drop vector in/out stalls
  - reduce the register loads for constants (the splat patterns)

Multiply 1

```
                        a3      a2      a1      a0
                        b3      b2      b1      b0
                    ----------------------------
                      a3.b0   a2.b0   a1.b0   a0.b0
              a3.b1   a2.b1   a1.b1   a0.b1
      a3.b2   a2.b2   a1.b2   a0.b2
a3.b3 a2.b3   a1.b3   a0.b3
-----------------------------------------------
```

Time tool…

```
0D      45                      ilhu        $9,32896
1D      456789                  stqd        $1,-112($1)
0D       56                     ai          $1,$1,-112
1D       567890                 lqa         $10,$CONSTANT_AREA+0
0D        67                    ilhu        $11,32896
1D        678901                lqa         $12,$CONSTANT_AREA+16
0D         78                   iohl        $9,1543
1D         7890                 cdd         $7,8($1)
0D          89                  iohl        $11,1029
1D          8901                cdd         $16,0($1)

0           78                                a           $3,$3,$5
0d          89                                a           $4,$4,$8
1d          --0123                            shlqbyi     $4,$4,4
0d             ---45                          a           $3,$3,$4
1d                --6789                      rotqbyi     $4,$3,8
1                    ---0123                  shufb     $2,$4,$7,$2
1                        ---456789            stqd        $2,0($6)
```

Other work

IBM numbers symmetric

Function		*1 SPE@ 3.2Ghz (Gbits/s)*	*8 SPE@ 3.2Ghz (Gbits/s)*	*Leading brand @ 3.2Ghz*
AES Encrypt (ECB)	128 bit key	2.059	16.472	1.029
	192 bit key	1.710	13.680	0.877
	256 bit key	1.462	11.696	0.762
AES Decrypt (ECB)	128 bit key	1.499	11.992	1.035
	192 bit key	1.252	10.016	0.870
	256 bit key	1.068		0.758
AES CTR	128 bit key	1.966	15.728	NA
	192 bit key	1.646	13.168	NA
	256 bit key	1.415	11.320	NA
DES (ECB)		0.492	3.936	0.427
TDES (ECB)		0.174	1.392	0.133
SHA-1		2.116	NA	0.902

Running DES on the Cell / Dag Arne Osvik

- Dag Arne Osvik @ SPEED

- Bitsliced implementation of DES
  - 128way parallelism per SPU
  - Sboxes optimized for SPU instruction set
- 4 Gbit/sec = 226 blocks/sec per SPU
- 32 Gbit/sec per Cell chip
- Can be used as a cryptographic accelerator (ECB, CTR, many CBC streams)

Breaking DES on the Cell / Dag Arne Osvik

- Reduce the DES encryption from 16 rounds to the equivalent of ~9.5 rounds, by shortcircuit evaluation and early aborts.
- Performance:
  - 108M=2^26.69 keys/sec per SPU
  - 864M=2^29.69 keys/sec per Cell chip

Comparison to FPGA

Expected time to break:

- COPACOBANA
  - ~9 days
  - €8,980
  - A year to build
- 52 PlayStation 3 consoles
  - ~9 days
  - €19,500 (at US$500 each)
  - Off the shelf

Running MD5 on the Cell / Dag Arne Osvik

- 32bit addition and rotation, boolean functions
- - Directly supported with 4waySIMD
- - Bitslice is slow: 128 adds require 94 instructions
- Many streams in parallel hide latencies
- Calculated compression function performance: Up to 15.6 Gbit/s per SPU

Running AES on the Cell / Dag Arne Osvik

- > 2.1 Gbit/s per SPU (~3.8 GHz Pentium 4)
- ~17 Gbit/s for full Cell, almost 13 Gbit/s for PS3
- CBC implementation only a little slower.

AES crypto analysis / Ken Roe

- Cell optimised AES took 35% of the time of x86 equivalent to search a key space.
- Price ratio at 3:1

Folding@home.

- **"Understand protein folding, misfolding, and related diseases"**
- "the PS3 takes the middle ground between GPU's (extreme speed, but at limited types of WU's) and CPU's (less speed, but more flexibility in types of WU's)."

Cell vs Rest FLOPS/Watt

	Cell	Intel clovertown	Amd winsor	Ati r590	Nvidea g80	PS2 EE	Ageia physx	PSP allegrex
Tech	90	65	65	90	90	250	130	90
Transistots	241	582	205	384	681	0	125	96
Chip area	235	296	127	352	480	80.8	189	96
Power (W)	110	120	92	200	177	6	20	0.35
Chip freq	3.2	2.7	3	0.65	1.35	0.3	350	333
Total SP GFLOPS	229	86.4	51.2	374	350	4.8	44.8	2.664
GFLOPS/W	2.08	0.72	0.56	1.87	1.98	0.80	2.24	7.61

Thank you !

Questions ?