Extended precision on the CELL processor

Diep Nguyen Hong Stef Graillat Jean-Luc Lamotte

LIP6/PEQUAN
P. and M. Curie University

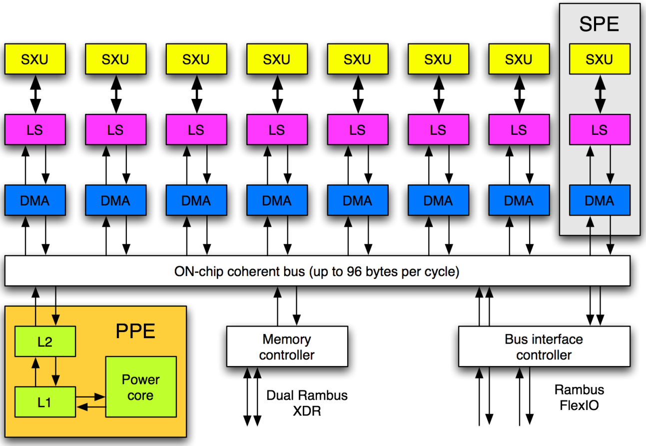
REC'08, Third International Worskhop on Reliable Engineering Computing

Savannah, Georgia, USA, February 20-22, 2008

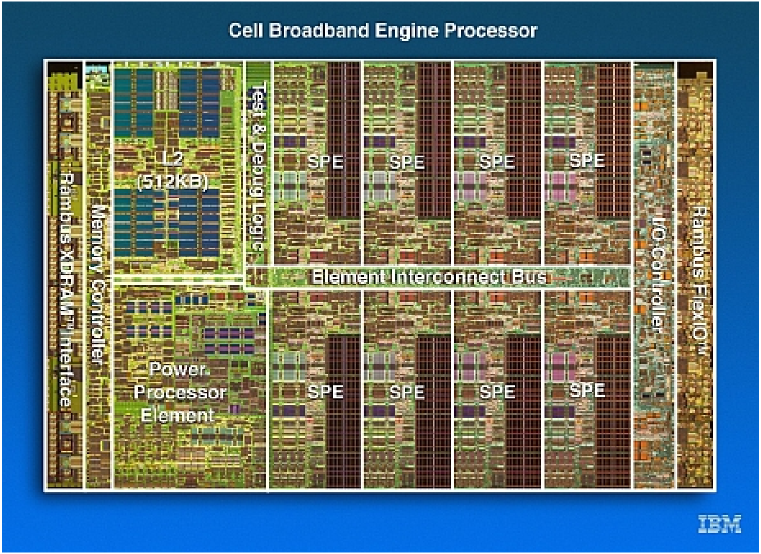
Overview

1. The Cell processor
2. Reliable computing and extended precision on Cell processor
3. Results
4. Conclusions

The CELL processor



SP > 200 GFlops, DP=15 Gflops, 25GB/s memory BW, 300 GB/s EIB


Cell Broadband Engine Processor
Rambus XDRAM™ Interface
Memory Controller
L2
(512KB)
Test & Debug Logic
SPE
SPE
SPE
SPE
Element Interconnect Bus
Power
Processor
Element
SPE
SPE
SPE
SPE
I/O Controller
Rambus FlexIO™
IBM

The PPE is based on the 2-way Power Architecture with :

- 32 KB of L1 cache for instructions
- 32 KB of L1 cache for data
- 512 KB of L2 cache

The PPE is fully pipelined for double precision computation and fully IEEE compliant.

Synergistic Processing Element SPE (1/2)

The SPE is a small processor with a vectorial unit.

- small memory (256 KB) for instructions and data, named "local store" (LS)
- 128 registers of 128 bits
- 1 SPU "Synergistic Processing Unit"
    - 4 units for single precision computation
    - 1 unit for double precision computation
- MFC "Memory Flow Controller" which manages memory access through DMA

Synergistic Processing Element SPE (2/2)

128-bit registers :

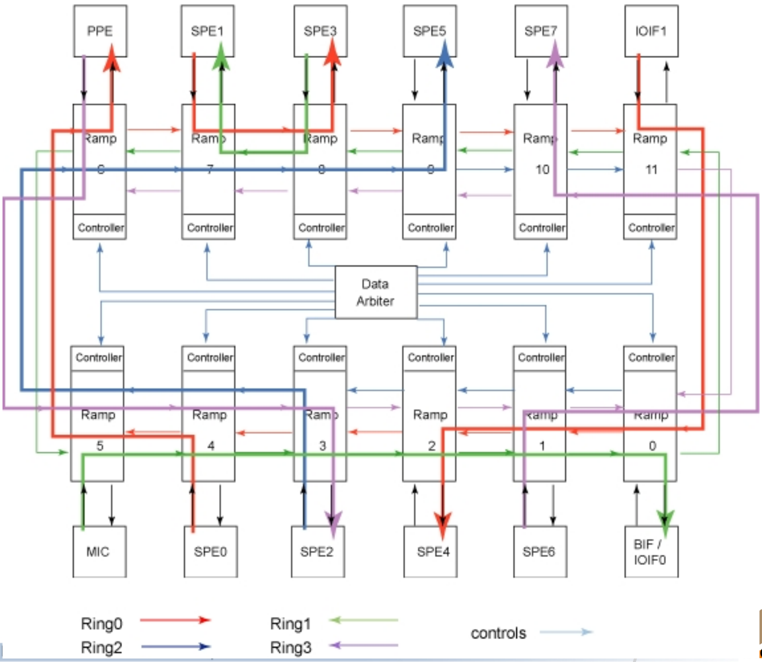
- 16 integers of 8-bits,
- 8 integers of 16-bits,
- 4 integers of 32-bits,
- 4 single precision floating point numbers,
- 2 double precision floating point numbers.

The SIMD processor is based on FMA and is fully pipelined in SP :

$$\text{Peak performance SP} : 4 \times 2 \times 3.2 = 25.6 GFLOPs$$

Not fully pipelined in double precision :

$$\text{Peak performance in DP} : 2 \times 2 \times 3.2/7 = 1.8 GFLOPs$$


PPE
SPE1
SPE3
SPE5
SPE7
IOIF1
Ramp 6
Ramp 7
Ramp 8
Ramp 9
Ramp 10
Ramp 11
Controller
Data Arbiter
Controller
Ramp 5
Ramp 4
Ramp 3
Ramp 2
Ramp 1
Ramp 0
MIC
SPE0
SPE2
SPE4
SPE6
BIF / IOIF0
Ring0
Ring2
Ring1
Ring3
controls

Parallelism on CELL

3 levels of parallelism :

1. processes on CELL processors, exchange with a MPI library,
2. threads on 8 SPE,
3. inside a thread, SIMD programming.

Parallelism on CELL

3 levels of parallelism :

1. processes on CELL processors, exchange with a MPI library,
2. threads on 8 SPE,
3. inside a thread, SIMD programming.

Parallelism on CELL

3 levels of parallelism :

1. processes on CELL processors, exchange with a MPI library,
2. threads on 8 SPE,
3. inside a thread, SIMD programming.

The parallel programming

- Data distribution and communication between PPE and SPE :
  - ALF
  - mailing box
  - exchange through DMA
  - data need to be aligned on quadword
  - double buffering technique
- on an SPE
  - only 256 KB
  - Altivec programming
  - code and data dependencies : not to break the SIMD pipeline

The performance price on SPE

No division

$1/x$ and $1/\sqrt{x}$: only the 12 first bits are exact.

SPU float arithmetic is not IEEE compliant :

- only rounding mode to zero (truncation).
- The highest exponent (128) is used not for Infinity or NaN, but is used to extend the range of the floating point.
- Inf and NaN are not recognized by arithmetic operations.
- Overflow results saturate to the largest representable positive or negative values, rather than producing $+/$-IEEE Infinity.
- No denormalized results : $+0$ instead.

The performance price

SPU double arithmetic is IEEE compliant except :

- FP trapping is not supported.
- Denormalized operands are treated as 0.
- NaN results are always the default QNaN (Quiet NaN)

Reliable computing on Cell processor

- difficult to implement interval arithmetic.
- possible to "emulate" a rounding mode toward $+\infty$
  if $r \in \mathbb{R}$ non-negative, $\mathrm{fl}_0(r) \le r \le \mathrm{succ}(\mathrm{fl}_0(r))$
  and
  $$\mathrm{succ}(f) = \max\{\mathrm{fl}_0(f + 2\mathbf{u}f), \mathrm{fl}_0(f + \underline{\mathrm{u}})\}.$$
  where $\mathbf{u}$ is the relative rounding error and $\underline{\mathrm{u}}$ the underflow unit

Error-free transformations

Let $a, b \in \mathbb{F}$, and $\circ$ an operation in $\circ \in \{+, -, \cdot, /\}$

$$\begin{aligned} (a \circ b) &\in \mathbb{R} \\ &\notin \mathbb{F} \\ \rightarrow fl(a \circ b) &\neq (a \circ b) \end{aligned}$$

$(a \circ b) - fl(a \circ b) = err$ is the roundoff error
"Error-free transformation" (EFT) : allows us to find the couple (x, y) such as :

- $x \approx fl(a \circ b)$
- $a \circ b = x + y$

EFT for the sum with rounding mode to nearest

$$x = \mathrm{fl}(a \pm b) \quad \Rightarrow \quad a \pm b = x + y \quad \text{with } y \in \mathbb{F},$$

Algorithm 1 (EFT for the sum of 2 floating point numbers (Knuth 1969))

function $[x, y] = \mathtt{TwoSum}(a, b)$
$\quad x = \mathrm{fl}(a + b)$
$\quad z = \mathrm{fl}(x - a)$
$\quad y = \mathrm{fl}((a - (x - z)) + (b - z))$

Cost : 6 FLOPs

Algorithm 2 (EFT for the sum of 2 floating point numbers (Dekker 1971), $|a| \geq |b|$)

function $[x, y] = \mathtt{FastTwoSum}(a, b)$
$\quad x = \mathrm{fl}(a + b)$
$\quad y = \mathrm{fl}((a - x) + b)$

Cost : 3 FLOPs

EFT for the sum with rounding mode toward zero

Algorithm 3 (EFT for the sum of 2 floating point numbers with a rounding mode toward zero (Priest))

function $[x, y] = \texttt{TwoSum} - \texttt{toward} - \texttt{zero}(a, b)$
 if $(|b| > |a|)$
  $\text{swap}(a, b)$
 $x = \text{fl}(a + b)$
 $d = fl(x - a)$
 $y = fl(b - d)$
 $\text{if}(y + d \neq b)$
  $x = a, y = b$

Cost : 6.5 FLOPs

EFT for the sum with rounding mode toward zero

Algorithm 4 (EFT for the sum of 2 floating point numbers with a rounding mode toward zero)

function $[x, y] = \texttt{TwoSum} - \texttt{toward} - \texttt{zero}(a, b)$
  if $(|b| > |a|)$
    swap(a, b)
  $x = \text{fl}(a + b)$
  $d = fl(x - a)$
  $y = fl(b - d)$
  $if(|2 * b| < |d|)$
    $x = a, y = b$

Cost : 6.5 FLOPs

Theorem 1

The algorithm $\texttt{TwoSum} - \texttt{toward} - \texttt{zero}$ *transforms 2 floating point numbers a and b into a couple of floating point numbers* (x, y) *satisfying*

$$x + y = a + b \text{ and } |y| < ulp(x)$$

EFT for the product with rounding mode to nearest

$$x = \mathrm{fl}(a \cdot b) \quad \Rightarrow \quad a \cdot b = x + y \quad \text{with } y \in \mathbb{F},$$

Algorithm `TwoProduct` of Veltkamp and Dekker (1971)

$$a = x + y \quad \text{and} \quad x \text{ and } y \text{ non-overlapping with } |y| \le |x|.$$

Algorithm 5 (Error-free split of a floating point number into two parts)

function $[x, y] = \texttt{Split}(a)$
  factor $= \mathrm{fl}(2^s + 1)$ % $\mathbf{u} = 2^{-p}$, $s = \lceil p/2 \rceil$
  $c = \mathrm{fl}(\texttt{factor} \cdot a)$
  $x = \mathrm{fl}(c - (c - a))$
  $y = \mathrm{fl}(a - x)$

Cost : 4 FLOPs

Algorithm 6 (EFT of the product of two floating point numbers)

function $[x, y] = \texttt{TwoProduct}(a, b)$

$x = \mathrm{fl}(a \cdot b)$

$[a_1, a_2] = \texttt{Split}(a)$

$[b_1, b_2] = \texttt{Split}(b)$

$y = \mathrm{fl}(a_2 \cdot b_2 - (((x - a_1 \cdot b_1) - a_2 \cdot b_1) - a_1 \cdot b_2))$

Cost : 17 FLOPs

EFT for the product with rounding mode to nearest

What is a Fused Multiply and Add ($\mathtt{FMA}$) in floating point arithmetic ?
$\longrightarrow$ Given a, b and c, three floating point numbers, $\mathtt{FMA}(a, b, c)$ computes $a \cdot b + c$ rounded according to the current rounding mode
$\Rightarrow$ only one rounding error for two operations !
$\mathtt{FMA}$ is available Cell processors.

Algorithm 7 (EFT of the product of two floating point numbers)

function $[x, y] = \mathtt{TwoProductFMA}(a, b)$
$x = \mathrm{fl}(a \cdot b)$
$y = \mathtt{FMA}(a, b, -x)$

$\Rightarrow$ Still valid with rounding toward zero !
Cost : 2 FLOPs

Extended precision

Definition 1 (extended precision)

An extended precision number of n is a non-evalued sum of n floating point number. $x = x_1 + x_2 + \ldots + x_n$

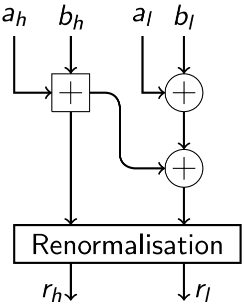
Normalisation :

1. to the nearest : $|x_{k+1}| \leq \frac{1}{2} ulp(x_k)$.
2. toward zero : $|x_{k+1}| < ulp(x_k)$ have the same sign.

Precision used on Cell processor : simple precision

- n=2 : double-simple

Sum of 2 double-simples



Theorem 2

Let $a = a_h + a_l$ and $b = b_h + b_l$, two double-simples to add, $r = r_h + r_l$ the result and δ the algorithm error. The algorithm error satisfies

$$r = a + b + \delta$$

$$|\delta| < max(2^{-23} * |a_l + b_l|, 2^{-43} * |a_h + a_l + b_h + b_l|) + 2^{-45} * |a + b|.$$

The exact transformation code

a, b : vector of 4 floating point numbers.

Code	cycles
TwoSum-toward-zero (a,b)	
comp = spu_cmpabsgt(b,a)	12
hi = spu_sel(a, b, comp)	-34
lo = spu_sel(b, a, comp)	45
s = spu_add(a , b)	012345
d = spu_sub(s , hi)	-678901
e = spu_sub(lo , d)	----234567
tmp = spu_mul(2 , lo)	789012
comp = spu_cmpabsgt(d, tmp)	34
s = spu_sel(s, hi, comp)	-56
e = spu_sel(e, lo, comp)	--89
return (s,e)	

Cost : 20 cycles

Renormalisation

```
Renormalise2-toward-zero (a,b)
    s = spu_add(a , b)
    comp = spu_cmpabsgt(b,a)
    hi = spu_sel(a, b, comp)
    lo = spu_sel(b, a, comp)
    d = spu_sub(s , hi)
    e = spu_sub(lo , d)
return (s,e)
```

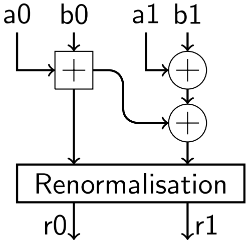
Cost : 18 cycles

Theorem 3

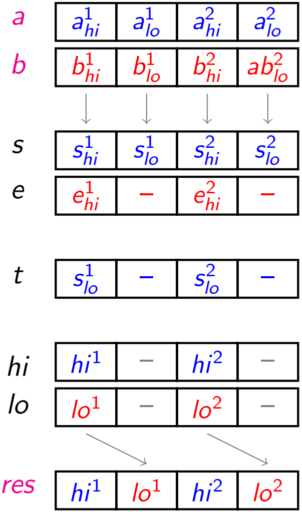
Let a and b be two single floating point numbers. The result returns by `Renormalise2-toward-zero` *is a double simple number (s, e) which satisfies*

- *s and e have the same sign*
- $|e| < ulp(s)$
- $a + b = s + e + \delta$ *with* $\delta \leq 2^{-45}|a + b|$.

Addition of two double-simple : the natural version




```
add_ds_ds_vect(a , b)
  (s,e) = TwoSum-toward-zero(a , b)
  t = spu_shuffle(s,s,switch-vect)
  t1 = spu_add(t , e)
  (hi,lo) = Renormalise2-toward-zero(s,t1)
  res = spu_shuffle(hi,lo,merge-vect)
return res
```



Cost : **50** cycles / **2** operations

addition of two double-simple : version 2

```
add_ds_ds_2vect (vect_a1, vect_a2, vect_b1, vect_b2)
    a_hi = spu_shuffle(vect_a1, vect_a2, _merge1_vect_)
    a_lo = spu_shuffle(vect_a1, vect_a2, _merge2_vect_)
    b_hi = spu_shuffle(vect_b1, vect_b2, _merge1_vect_)
    b_lo = spu_shuffle(vect_b1, vect_b2, _merge2_vect_)
    (s, e) = TwoSum-toward-zero (a_hi, b_hi)
    t1 = spu_add(a_lo , b_lo)
    tmp = spu_add(t1 , e)
    (hi, lo) = Renormalise2-toward-zero (s , tmp)
    vect_c1 = spu_shuffle(hi, lo, _merge1_vect_)
    vect_c2 = spu_shuffle(hi, lo, _merge2_vect_)
return (vect_c1, vect_c2)
```

Cost : **64** cycles / **4** opérations

Sum of double-simple : optimised version

- The version 2 increases the performance of the sum.
- cycles are still lost.

$\Rightarrow$ to perform version 2 twice in a same function.

Cost : **72** cycles / **8** operations

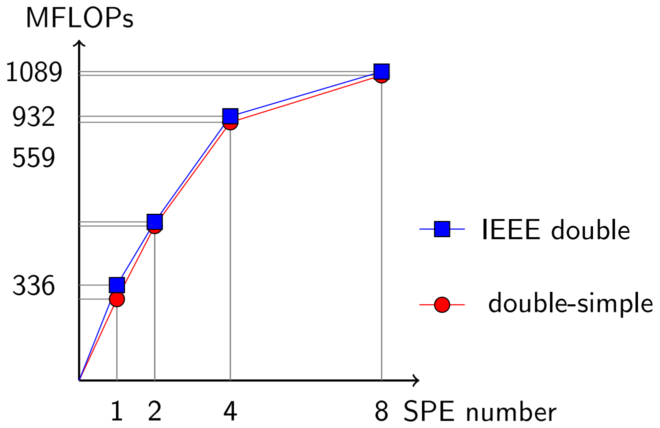
Theoretical results

frequency : 3.2GHz.
The peak performance in double precision :
$2 \times 2 \times 3.2/7 = 1.8 GFLOPs$.

function	**Cycles number**	**Performance**
Add_ds_ds_vect	50 cycles / 2	128 MFLOPs
Add_ds_ds_2vect	64 cycles / 4	200 MFLOPs
Add_ds_ds_4vect	72 cycles / 8	355 MFLOPs
Mul_ds_ds_vect	49 cycles / 2	130 MFLOPs
Mul_ds_ds_2vect	60 cycles / 4	213 MFLOPs
Mul_ds_ds_4vect	63 cycles / 8	406 MFLOPs

Comparison with double precision

Addition of two vectors with DMA to load data on SPE



Functions	**Theoretical (1SPE)**	**Measured (1 SPE)**	**Measured (8 SPEs)**
Add_ds_ds_4vect	355	266	2133
Mul_ds_ds_4vect	406	320	2560
double precision addition	914	914	7314
double precision product	914	914	7314

TAB.: Performance without data exchange (MFLOPS)

Quad simple

function	Cycles number	Performance
Add_qs_qs_4vect	449 cycles / 4	28.5 MFLOPs
Mul_qs_qs_4vect	583 cycles / 4	21.9 MFLOPs

Conclusions

The true goal :

- to prepare the work for the next CELL :
  - fully pipelined double precision floating point number
  - probably 512 KB on SPE
- a double double library,
- a quad double library.

Rumours on the next generation

- IEEE compliant
- from 8 to 32 SPE
- over 1TFLOPS

Références I

Yozo Hida, Xiaoye S.Li, and David H.Baily.
Quad-double arithmetic : Algorithms, implementations, and application.
2000.

Donald Knuth.
The art of computer programming : Seminumerical algorithms
vol.2, Reading, Massachusetts : Addison-Wesley,2000.

Takeshi Ogita, Siegfried M. Rump, and Shin'ichi Oishi.
Accurate sum and dot product.
SIAM J. Sci. Comput., 26(6) :1955–1988, 2005.

Christoph Quirin Lauter.
Basic building blocks for a triple-double intermediate format
Tech. report, INRIA, 2005

Références II

Douglas M. Priest.
On Properties of Floating Point Arithmetics : Numerical Stability and the Cost of Accurate Computations.
PhD thesis, Mathematics Department, University of California, Berkeley, CA, USA, November 1992.

T.J Dekker.
A floating-point technique for extending the available precision.
Numer . Math., 18 : 224–242, 1971.