Numerical Accuracy and Reliability Issues in HPC

SIAM CSE, Boston (USA), February 25th, 2013

Towards a Reliable Performance Evaluation of Accurate Summation Algorithms

Philippe Langlois, Bernard Goossens, David Parello

University of Perpignan Via Domitia, DALI,
University Montpellier 2, LIRMM,
CNRS UMR 5506, France

UPVD
Université de Perpignan Via Domitia

DALI
Digits, Architectures et Logiciels Informatiques

How to manage accuracy and speed?

A new "better" algorithm every year since 1999

1965 Møller, Ross
1969 Babuska, Knuth
1970 Nickel
1971 Dekker, Malcolm
1972 Kahan, Pichat
1974 Neumaier
1975 Kulisch/Bohlender
1977 Bohlender, Mosteller/Tukey
1981 Linnaimaa
1982 Leuprecht/Oberaigner
1983 Jankowski/Semoktunowicz/-Wozniakowski
1985 Jankowski/Wozniakowski
1987 Kahan
1991 Priest
1992 Clarkson, Priest
1993 Higham
1997 Shewchuk
1999 Anderson
2001 Hlavacs/Uberhuber
2002 Li et al. (XBLAS)
2003 Demmel/Hida, Nievergelt, Zielke/Drygalla
2005 Ogita/Rump/Oishi, Zhu/Yong/Zeng
2006 Zhu/Hayes
2008 Rump/Ogita/Oishi
2009 Rump, Zhu/Hayes
2010 Zhu/Hayes

Accurate or faithful floating point summation

Limited accuracy for backward stable sums

- Accuracy of the computed sum $\leq (n-1) \times \mathit{cond} \times \mathbf{u}$
- No more significant digit in IEEE-b64 for large cond, *i.e.* $> 10^{16}$

Accurate but still conditioning dependent

- Accuracy of the computed sum $\lesssim \mathbf{u} + \mathit{cond} \times \mathbf{u}^K$
- double-double, compensated sums: Kahan(72), Sum2(05), SumK(05)

Faithfully or correctly rounded sums

- Accuracy of the computed sum $\leq \mathbf{u}$
- Kahan (87), . . . , Rump *et al.*: AccSum (SISC-08), FastAccSum (SISC-09)
  Zhu-Hayes: iFastSum, HybridSum (SISC-09), OnLineExact (TOMS-10)

Accurate or faithful floating point summation

Limited accuracy for backward stable sums

- Accuracy of the computed sum $\leq (n-1) \times \mathit{cond} \times \mathbf{u}$
- No more significant digit in IEEE-b64 for large cond, *i.e.* $> 10^{16}$

Accurate but still conditioning dependent

- Accuracy of the computed sum $\lesssim \mathbf{u} + \mathit{cond} \times \mathbf{u}^K$
- double-double, compensated sums: Kahan(72), Sum2(05), SumK(05)

Faithfully or correctly rounded sums

- Accuracy of the computed sum $\leq \mathbf{u}$
- Kahan (87), . . . , Rump *et al.*: AccSum (SISC-08), FastAccSum (SISC-09)
  Zhu-Hayes: iFastSum, HybridSum (SISC-09), OnLineExact (TOMS-10)
- Run-time and memory efficiencies are now the choice factors

Reliable and significant measure of the time complexity?

Flop count vs. run-time measures: which one trust?

Metric	Sum	DDSum	Sum2
Flop count	$n-1$	$10n$	$7n$
Flop count ratio vs. Sum (approx.)	1	10	7
Measured #cycles ratio (approx.)	1	7.5	2.5

- Flop counts and measured run-times are not proportional
- Run-time measure is a very difficult experimental process

How to trust non-reproducible experiment results?

Measures are mostly non-reproducible

- The execution time of a binary program varies, even using the same data input and the same execution environment.

Why? Experimental uncertainty (even) of the hardware performance counters

- Spoiling events: background tasks, concurrent jobs, OS interrupts
- Non predictable issues: instruction schedul., branch pred., cache mng.
- Timing in seconds depends on external conditions: temperature of the room
- Timing in cycles difficult: 1 core cycle $\neq$ 1 bus cycle on modern processors

Uncertainty increases as computer system complexity does

- Architecture and micro-architecture issues: multicore, hybrid, speculation
- Compiler options and its effects

Software and system performance experts' point of view

The limited *Accuracy of Performance Counter Measurements*

> *We caution performance analysts to be suspicious of cycle counts . . . gathered with performance counters.*
>
> *D. Zaparanuks, M. Jovic, M. Hauswirth (2009)*

Can Hardware Performance Counters Produces Expected, Deterministic Results?

> *In practice counters that should be deterministic show variation from run to run on the x86_64 architecture. . . . it is difficult to determine known "good" reference counts for comparison.*
>
> *V.M. Weaver, J. Dongarra (2010)*

How to trust the current literature?

Numerical results in S.M. Rump et al. contributions (for summation)

- 26% for Sum2-SumK (SISC-05) : 9 pages over 34
- 20% for AccSum (SISC-08) : 7 pages over 35
- 20% for AccSumK-NearSum (SISC-08b) : 6 pages over 30
- **less that 3%** for FastAccSum (SISC-09) : 1 page over 37

Lack of proof, or at least of reproducibility

> *Measuring the computing time of summation algorithms in a high-level language on today's architectures is more of a hazard than scientific research.* *S.M. Rump (SISC, 2009)*

. . . in the paper entitled *Ultimately Fast Accurate Summation*

Outline

ILP and the performance potential of the algorithm

Instruction Level Parallelism (ILP) describes the potential of the instructions of a program that can be executed simultaneously

Hennessy-Patterson's ideal machine (H-P IM)

- every instruction is executed one cycle after the execution one of the producers it depends
- no other constraint than the true instruction dependency (RAW)

Our ideal run measures : C=#cycles, I=# instruc. and I/C

- ideal run = maximal exploitation of the program ILP
- ILP measures the potential of the algorithm performance
- processor and ILP in practice: superscalar out-of-order executions

The ideal execution of Sum: hand-made analysis

The ideal execution of Sum takes n cycles

Sum		iter.	1	2	3	...	$n-1$
	`s = x[0];`		0				
	`for(i=1; i<n; i++)`						
a	`  s = s + x[i];`		1	2	3	$\cdots$	n-1
	`return(s);`						n

No ILP in Sum

- $C_{\text{Sum}} = n$
- $I = n$
- ILP=1

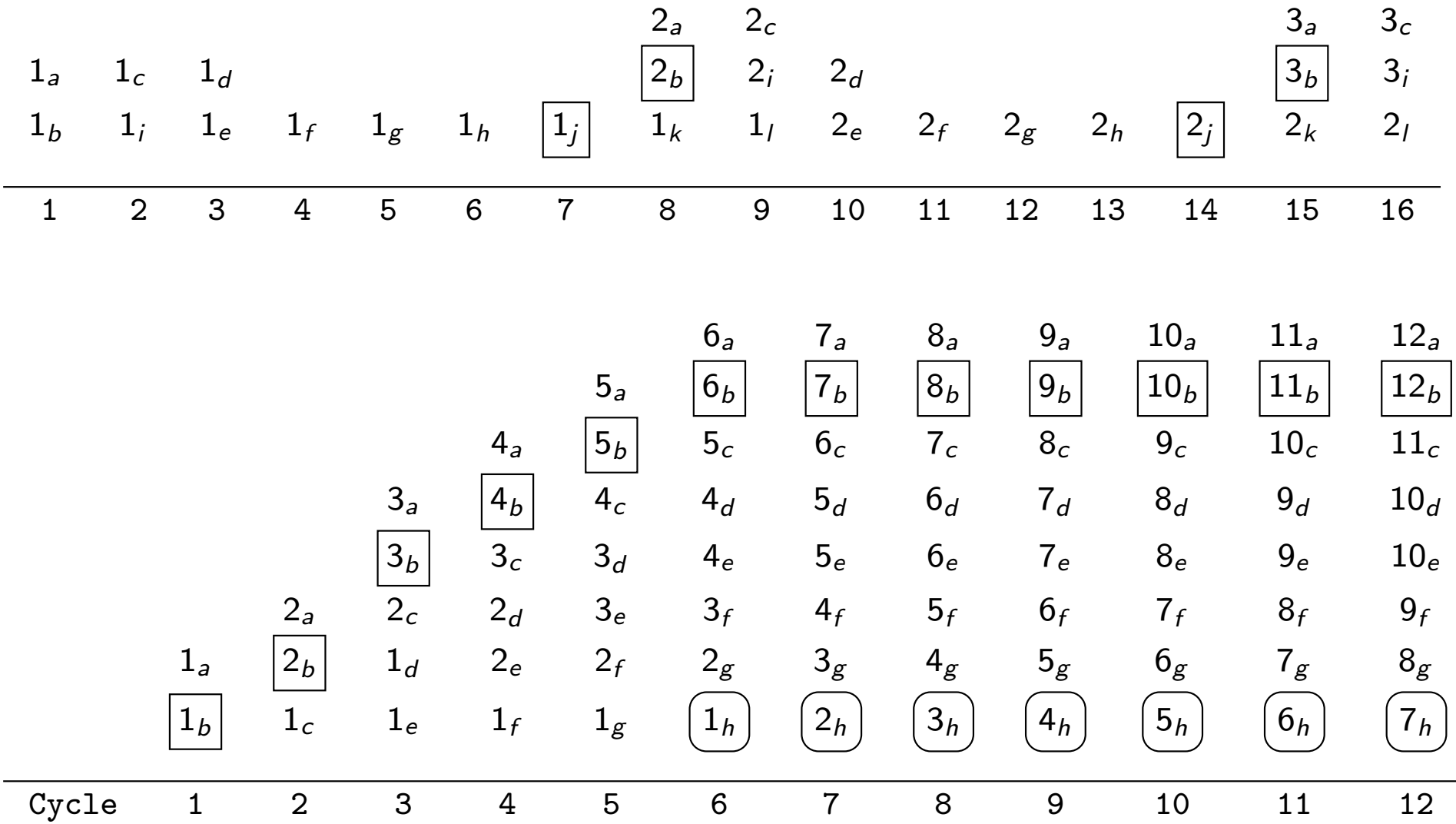
DDSum ideally runs in $7n - 5$ cycles

DDSum		iter.	1	2	3	...	$n-1$
	`s = x[0];`		0				
	`for(i=1; i<n; i++){`						
a	`  s_ = s;`		1	8	15	···	7n-13
b	`  s = s + x[i];`		1	8	15	···	7n-13
c	`  t = s - s_;`		2	9	16	···	7n-12
d	`  t2 = s - t ;`		3	10	17	···	7n-11
e	`  t3 = x[i] - t;`		3	10	17	···	7n-11
f	`  t4 = s_ - t2;`		4	11	18	···	7n-10
g	`  t5 = t4 + t3;`		5	12	19	···	7n-9
h	`  s_l = s_l + t5;`		6	13	20	···	7n-8
i	`  s_ = s;`		2	9	16	···	7n-12
j	`  s = s + s_l;`		7	14	21	···	7n-7
k	`  e = s_ - s;`		8	15	22	···	7n-6
l	`  s_l = s_l + e;`		9	16	23	···	7n-5
	`}`						
	`return(s);`						7n-4

Sum2 ideally runs in $n + 7$ cycles

Sum2		iter.	1	2	3	...	$n-1$
	`s = x[0];`		0				
	`for(i=1; i<n; i++){`						
a	`  s_ = s;`		1	2	3	···	n-1
b	`  s = s + x[i];`		1	2	3	···	n-1
c	`  t = s - s_;`		2	3	4	···	n
d	`  t2 = s - t ;`		3	4	5	···	n+1
e	`  t3 = x[i] - t;`		3	4	5	···	n+1
f	`  t4 = s_ - t2;`		4	5	6	···	n+2
g	`  t5 = t4 + t3;`		5	6	7	···	n+3
h	`  c = c + t5;`		6	7	8	···	n+4
	`}`						
	`return(s+c);`						n+6

Less ILP in DDSum(top) than in Sum2 (bottom)



ILP hand-made analysis: conclusion

Metric	Sum	DDSum	Sum2
Flop count (approx. ratio)	1	10	7
Measured #cycles (approx. ratio)	1	7.5	2.5
Flop count / measured #cycles (approx.)	1	1.4	2.8
Ideal C (approx. ratio)	1	7	1
Ideal flop count / C (approx.)	1	1.7	8

- DDSum actually run as fast as it can
- Current architectures exploit only 30% of Sum2's ILP
- Huge potential in Sum2 which can run as fast as Sum

The PerPI Tool automatizes this ILP analysis

PerPI: a pintool to analyse and visualise the ILP of x86-coded algorithms

- Pin (Intel) tool (http://www.pintool.org)
- Outputs: ILP measure (#C, #I), IPC histogram, data-dependency graph
- Input: x86_64 binary file
- Developed and maintained by B. Goossens and D. Parello (DALI)
- In progress: `http://perso.univ-perp.fr/david.parello/perpi/`

Seven recent accurate and fast summation algorithms

Recursive summation (not accurate)

- Sum

Accurate sums: twice more precision

- Sum2
- DDSum

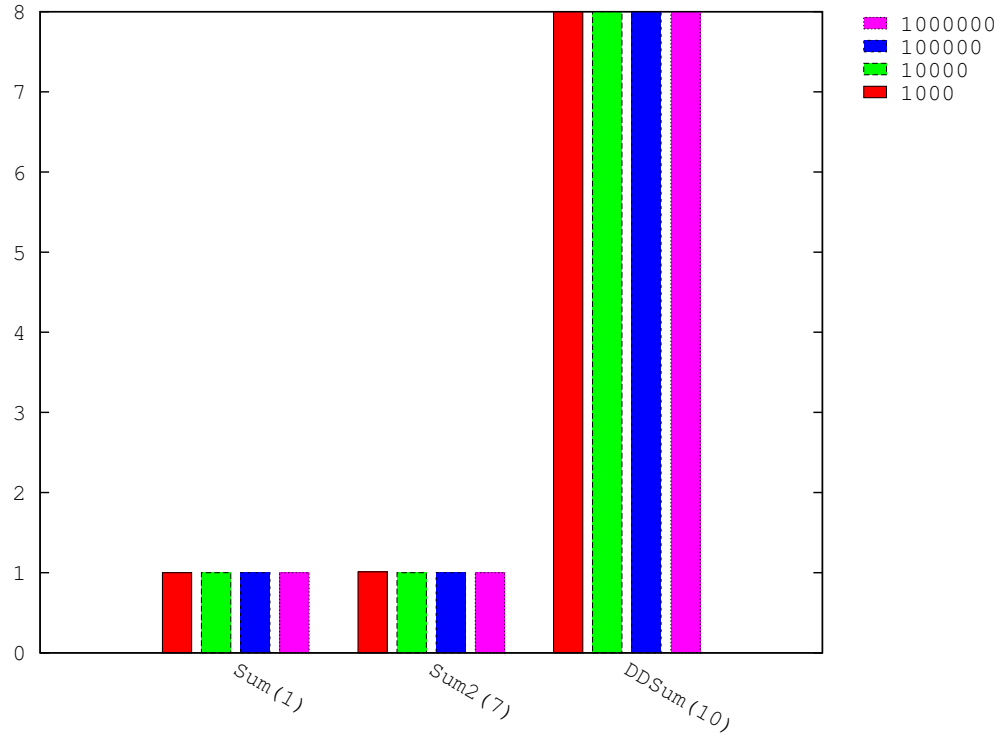
Faithfully or exactly rounded sums

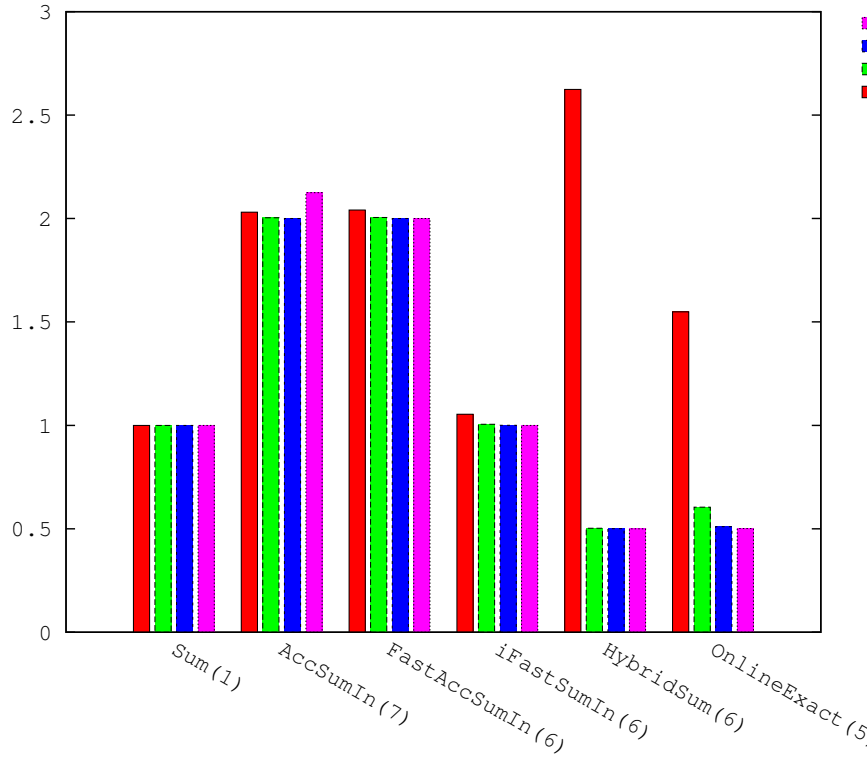
- iFastSum
- AccSum
- FastAccSum
- HybridSum
- OnLineExactSum

PerPI and reproducibility: one run is enough

```
start : <main>   (depth: 2, lcid: 104)
  stop  : <Sum>   (depth: 3, lcid: 10201)(cid: 10201)    I[13781]::C[10000]::ILP[1.3781]
  stop  : <Sum>   (depth: 3, lcid: 10203)(cid: 10203)    I[13781]::C[10000]::ILP[1.3781]
  stop  : <Sum>   (depth: 3, lcid: 10205)(cid: 10205)    I[13781]::C[10000]::ILP[1.3781]
  stop  : <iFastSumIn>   (depth: 3, lcid: 10207)(cid: 10207)    I[696088]::C[18043]::ILP[38.5794]
  stop  : <iFastSumIn>   (depth: 3, lcid: 10241)(cid: 10241)    I[696076]::C[18043]::ILP[38.5787]
  stop  : <iFastSumIn>   (depth: 3, lcid: 10275)(cid: 10275)    I[696076]::C[18043]::ILP[38.5787]
  start : <OnlineExactSum>   (depth: 3, lcid: 10309)
    stop  : <iFastSumIn>   (depth: 4, lcid: 10320)(cid: 10320)    I[29704]::C[611]::ILP[48.6154]
  stop  : <OnlineExactSum>   (depth: 3, lcid: 10309)(cid: 10309)    I[301467]::C[10607]::ILP[28.4215]
 stop  : <main>   (depth: 2, lcid: 104)(cid: 104)    I[2884900]::C[49320]::ILP[58.4935]
Global ILP (cid: 0)    I[2895541]::C[49572]::ILP[58.4108]
```

PerPI # cycle ratio: accurate (left) and faithful sums (right)

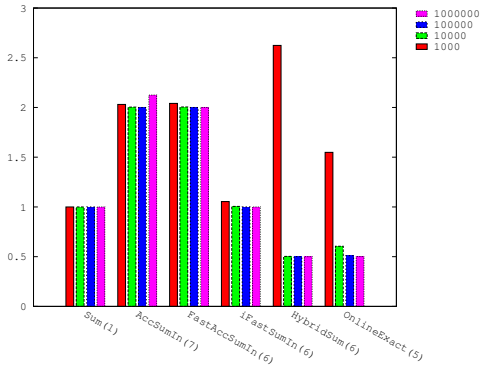




- $cond = 10^{32}$ and $n = 10^3, 10^4, 10^5, 10^6$.
- Twice more precision "free" with the compensated sum
- Faithful sum for even less

How to trust it? PerPI bug vs. reality?

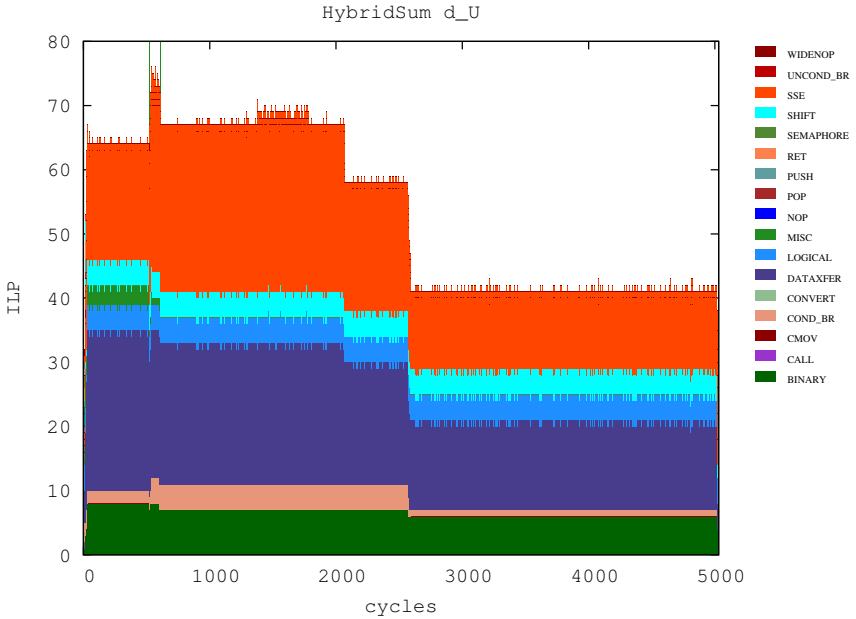
Huge ILP of HybridSum and OnLineExact

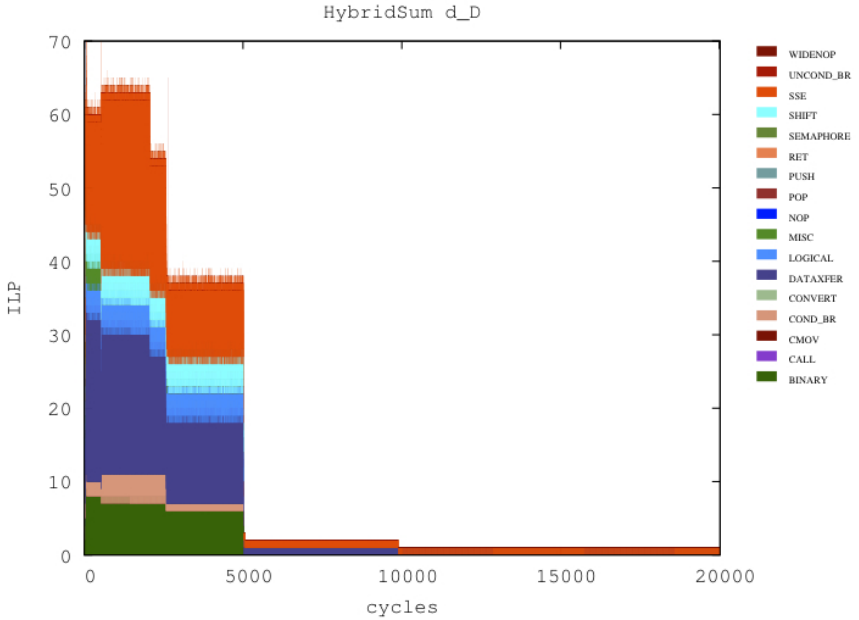


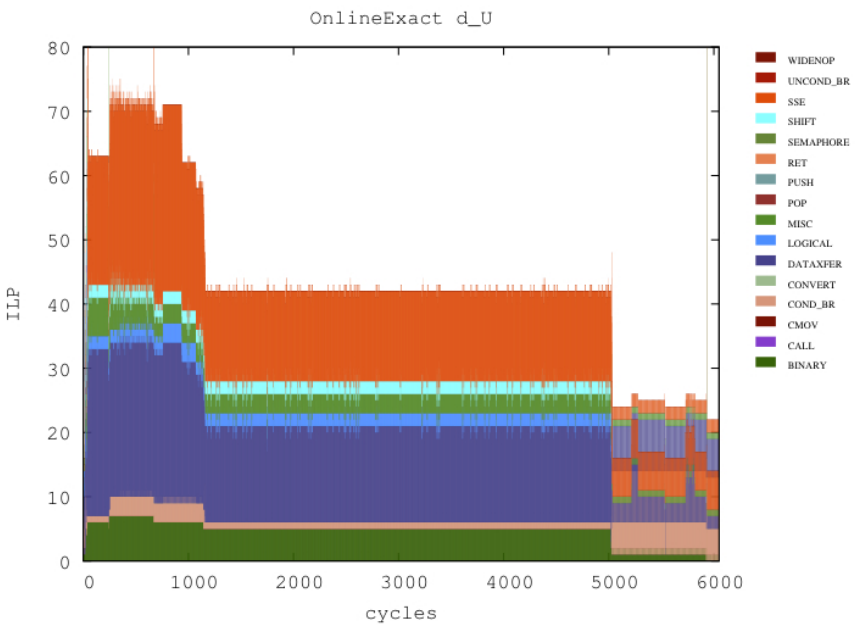
PerPI helps to highlight many details . . . but not all

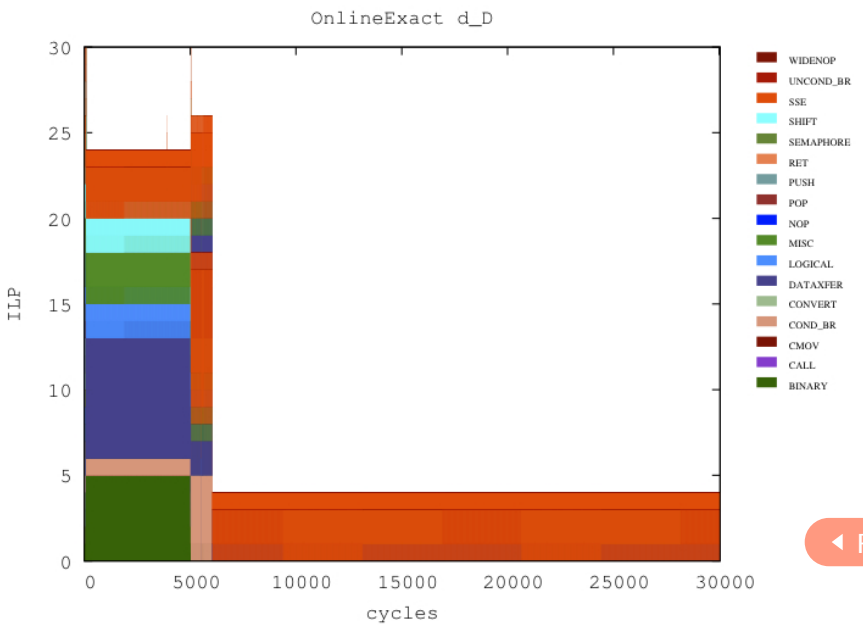
- PerPI measures and exhibits C=$n/2$ ▸ histograms
- Assembly code analysis confirms C=$n/2$
- Floating-point consistency: Sum does not benefit from loop unrolling
- Faithfulness consistency: use as much as possible optimisations
- Loop unroll ($\times 2$) in the exponent extraction step
- Short vector summation starts as soon as possible . . .
- . . . depending on the distribution of the data exponent even for a given exponent range ▸▸ Skip ▸ x86 peculiarities

PerPI histograms for HS (↑) and OLE (↓) for d_U: left, d_D: right



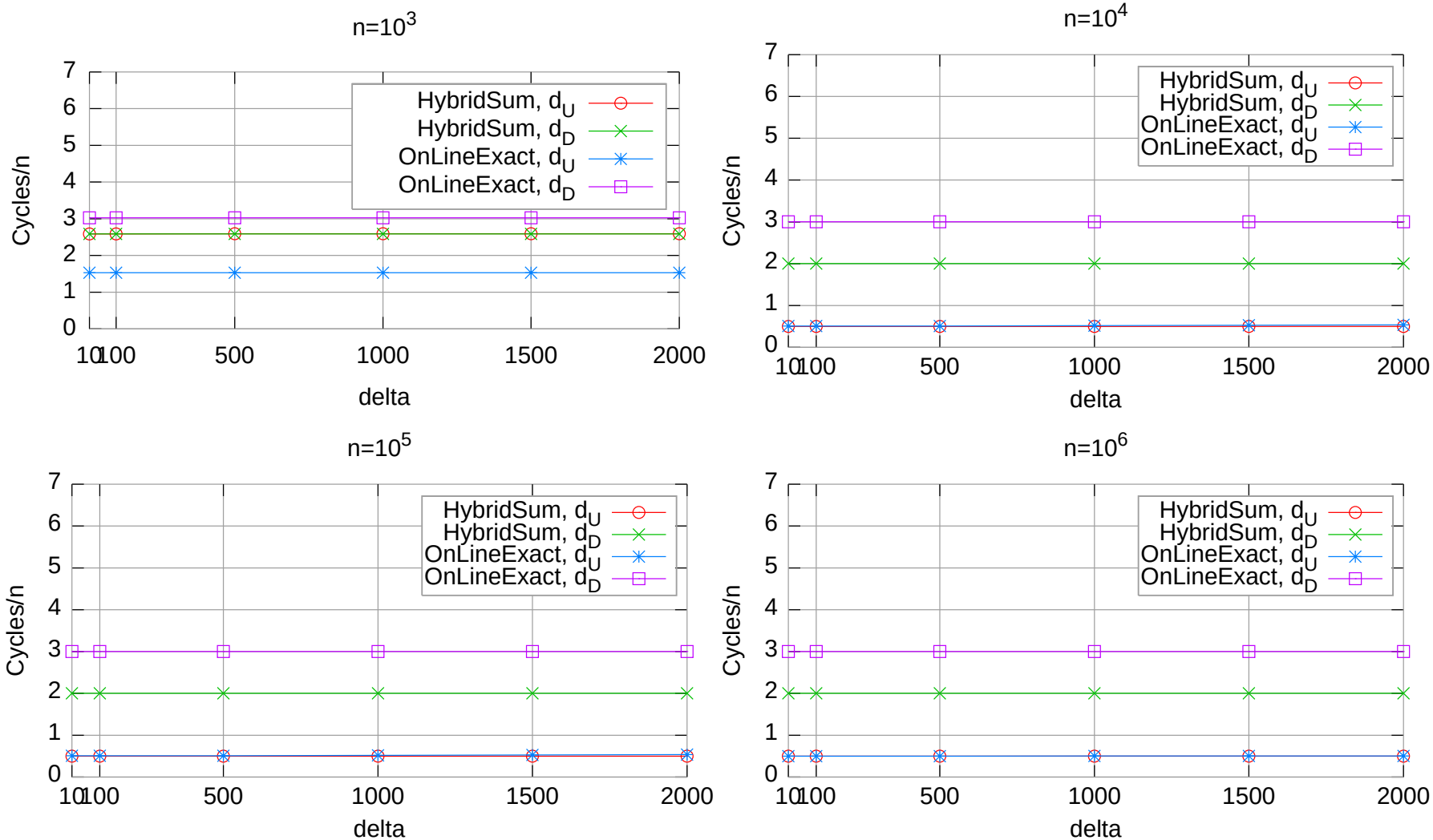






◀ Ret.

Exponent distribution at $\delta = cst$ for HS and OLE



d_U: uniform dist. in $[-\delta/2, \delta/2]$ vs. d_D: Dirac-like distr.: one $-\delta/2$, n-1 $\delta/2$

Zooming and understanding the worst measures (d_D)

```
start : <HybridSum>
  start : <iFastSumIn>
  stop  : <iFastSumIn>      I[62719]::C[2580]::ILP[24.3097]
stop  : <HybridSum>       I[267980]::C[20020]::ILP[13.3856]
start : <OnlineExact>
  start : <iFastSumIn>
  stop  : <iFastSumIn>      I[334]::C[32]::ILP[10.4375]
stop  : <OnlineExact>     I[229263]::C[30026]::ILP[7.63548]
```

Explanation: extraction step and x86 ISA peculiarities

- Cycles between two iterations: 2 in HS vs. 3 in OLE

Conclusion

Conclusion

- Highly accurate algorithm needs reliable performance evaluation
- PerPI provides reproducible measures of the performance potential
- PerPI highlights how the algorithm and the architecture interact
- PerPI may help to improve the algorithm or its implementation
- Hand-made vs. PerPI analysis: the ideal machine vs. one ISA machine
- Towards a dynamic reference repository for accurate sums and other core routines

▸ exponent distribution ▸ x86 peculiarities

References I

D. H. Bailey.

Twelve ways to fool the masses when giving performance results on parallel computers.

Supercomputing Review, pages 54–55, Aug. 1991.

B. Goossens, P. Langlois, D. Parello, and E. Petit.

PerPI: A tool to measure instruction level parallelism.

In K. Jónasson, editor, *Applied Parallel and Scientific Computing - 10th International Conference, PARA 2010, Reykjavík, Iceland, June 6-9, 2010, Revised Selected Papers, Part I*, volume 7133 of *Lecture Notes in Computer Science*, pages 270–281. Springer, 2012.

N. J. Higham.

Accuracy and Stability of Numerical Algorithms.

SIAM, 2nd edition, 2002.

J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, and S. Torres.

Handbook of Floating-Point Arithmetic.

Birkhäuser Boston, 2010.

References II

T. Ogita, S. M. Rump, and S. Oishi.

Accurate sum and dot product.

SIAM J. Sci. Comput., 26(6):1955–1988, 2005.

S. M. Rump.

Ultimately fast accurate summation.

SIAM J. Sci. Comput., 31(5):3466–3502, 2009.

S. M. Rump, T. Ogita, and S. Oishi.

Accurate floating-point summation – part I: Faithful rounding.

SIAM J. Sci. Comput., 31(1):189–224, 2008.

V. Weaver and J. Dongarra.

Can hardware performance counters produce expected, deterministic results?

In *3rd Workshop on Functionality of Hardware Performance Monitoring, 2010*, pages 1–11, Atlanta, USA, 2010.

References III

D. Zaparanuks, M. Jovic, and M. Hauswirth.

Accuracy of performance counter measurements.

In *IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2009, April 26-28, 2009, Boston, Massachusetts, USA*, pages 23–32, 2009.

Y.-K. Zhu and W. B. Hayes.

Correct rounding and hybrid approach to exact floating-point summation.

SIAM J. Sci. Comput., 31(4):2981–3001, 2009.

Y.-K. Zhu and W. B. Hayes.

Algorithm 908: Online exact summation of floating-point streams.

ACM Transactions on Mathematical Software, 37(3):37:1–37:13, Sept. 2010.