High Intensity Matrix Polynomial Solvers for the Heat and Poisson Equations.

Thierry Dumont

Intitut Camille Jordan, Lyon.

Maison de la Simulation, 11 Avril 2017.

- Collaboration with A. Darte (ENSL), T. Guillet (Intel), V. Louvet (ICJ), R. Prat (ICJ),
- Motivated by a long time common work with S. Descombes (Nice), M. Massot (ECP), M. Duarte (ECP)...

Sketch of the talk

1. Motivation: a Reaction–Diffusion–Convection Problem.
2. A short reminder about the Roofline Model.
3. Runge–Kutta methods and linear problems.
4. Stabilized RK methods.
5. Stabilized RK methods $+$ Discontinuous Galerkin $=>$ performances.
6. The Poisson equation.

Positive Streamer Simulations

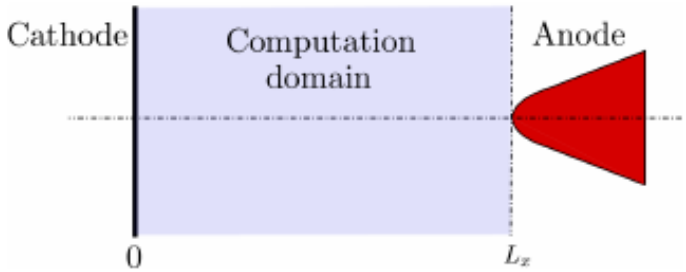


Figure 1: Computational domain for the studied point-to-plane geometry.

Repetitive periodic discharges: a high voltage pulse is applied during $\simeq 10^{-8}$ second, followed by a relaxation of $\simeq 10^{-4}$ second. Distance between cathode and anode: 1 cm.

Drift-Diffusion Equations:

$$
\begin{aligned}
\partial_t n_\mathrm{e} - \partial_x \cdot n_\mathrm{e}\,\mathbf{v}_\mathrm{e} - \partial_x \cdot (D_\mathrm{e}\,\partial_x n_\mathrm{e}) &= n_\mathrm{e}\alpha|\mathbf{v}_\mathrm{e}| - n_\mathrm{e}\eta|\mathbf{v}_\mathrm{e}| + n_\mathrm{e} n_\mathrm{p}\beta_\mathrm{ep} + n_\mathrm{n}\gamma \\
\partial_t n_\mathrm{p} + \partial_x \cdot n_\mathrm{p}\mathbf{v}_\mathrm{p} - \partial_x \cdot (D_\mathrm{p}\,\partial_x n_\mathrm{p}) &= n_\mathrm{e}\alpha|\mathbf{v}_\mathrm{e}| - n_\mathrm{e} n_\mathrm{p}\beta_\mathrm{ep} + n_\mathrm{n} n_\mathrm{p}\beta_\mathrm{np} \\
\partial_t n_\mathrm{n} - \partial_x \cdot n_\mathrm{n}\mathbf{v}_\mathrm{n} - \partial_x \cdot (D_\mathrm{n}\,\partial_x n_\mathrm{n}) &= n_\mathrm{e}\eta|\mathbf{v}_\mathrm{e}| - n_\mathrm{n} n_\mathrm{p}\beta_\mathrm{np} - n_\mathrm{n}\gamma
\end{aligned}
$$

$$
\varepsilon_0\,\partial_x^2 V = -q_\mathrm{e}(n_\mathrm{p} - n_\mathrm{n} - n_\mathrm{e}), \qquad \mathbf{E} = -\partial_x V, \qquad \mathbf{v}_i = \mu_i \mathbf{E}.
$$

Duarte et al.: *A new numerical strategy with space-time adaptivity and error control for multi-scale streamer discharge simulations* – JCP.,2012.

Positive Streamer Simulations

Fastest time scales in reaction: $\simeq 10^{-14}$ second. This is a true multiscale problem!

Positive Streamer Simulations

Fastest time scales in reaction: $\simeq 10^{-14}$ second. This is a true multiscale problem!

Strang Splitting:

$$\mathcal{S}_{\Delta t}(u_0) = \mathcal{R}_{\Delta t/2}\mathcal{D}_{\Delta t/2}\mathcal{C}_{\Delta t}\mathcal{D}_{\Delta t/2}\mathcal{R}_{\Delta t/2}(u_0).$$

Positive Streamer Simulations

Fastest time scales in reaction: $\simeq 10^{-14}$ second. This is a true multiscale problem!
Strang Splitting:

$$\mathcal{S}_{\Delta t}(u_0) = \mathcal{R}_{\Delta t/2}\mathcal{D}_{\Delta t/2}\mathcal{C}_{\Delta t}\mathcal{D}_{\Delta t/2}\mathcal{R}_{\Delta t/2}(u_0).$$

Choice of $\mathbf{v}_i$ in $\mathcal{C}_{\Delta t}$?

1. Compute $\mathbf{v}_i$ from u_0 (solve a Poisson equation) $=>$ method of order 1.

Positive Streamer Simulations

Fastest time scales in reaction: $\simeq 10^{-14}$ second. This is a true multiscale problem!

Strang Splitting:

$$\mathcal{S}_{\Delta t}(u_0) = \mathcal{R}_{\Delta t/2}\mathcal{D}_{\Delta t/2}\mathcal{C}_{\Delta t}\mathcal{D}_{\Delta t/2}\mathcal{R}_{\Delta t/2}(u_0).$$

Choice of $\mathbf{v}_i$ in $\mathcal{C}_{\Delta t}$?

1. Compute $\mathbf{v}_i$ from u_0 (solve a Poisson equation) $=>$ method of order 1.
2. Compute $u_{1/2} = \mathcal{C}_{\Delta t/2}\mathcal{D}_{\Delta t/2}\mathcal{R}_{\Delta t/2}(u_0)$, and compute $\mathbf{v}_i$ from $u_{1/2}$ (solve a Poisson equation) $=>$ method of order 2.

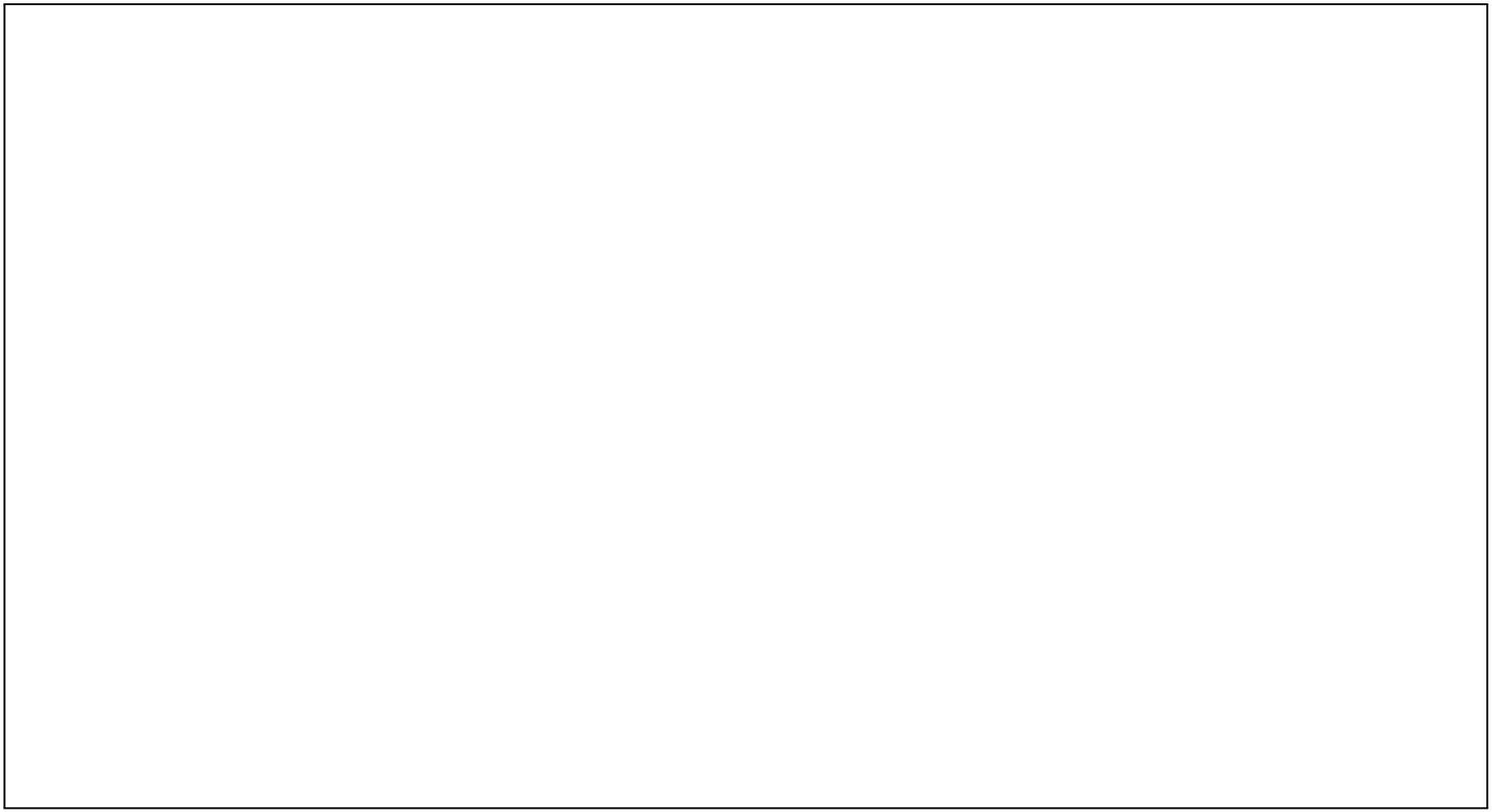
Positive Streamer Simulations

- We have approximations at order 1 and 2 $=>$ time step adaptation.

Positive Streamer Simulations

- We have approximations at order 1 and 2 $=>$ time step adaptation.
- Solution has large propagating gradients: we need to adapt the mesh. We use a multiresolution strategy do adapt mesh after each step (we use finite volumes).

Positive Streamer Simulations



Positive Streamer Simulations

At each time step:

- solve 2 Poisson equations.
- solve 4 heat equations.

At each time step, Δt and the mesh change and thus "matrices" change.

Positive Streamer Simulations

- Solution of Poisson equation, and solution of the Heat equation with implicit methods:
  - (Incomplete) factorisation of matrices are expensive.
  - Solution is expensive.
  - The largest part the total computing time in dimension 2, whatever the method we use.

Positive Streamer Simulations

- Solution of Poisson equation, and solution of the Heat equation with implicit methods:
  - (Incomplete) factorisation of matrices are expensive.
  - Solution is expensive.
  - The largest part the total computing time in dimension 2, whatever the method we use.

Computers have change a lot since the begining of my career!

Computers have change a lot since the begining of my career!

A short reminder about the Roofline Model.

- SIMD:

AVX instructions:

In one clock cycle, do:

$y_i = a_i\, x_i + b_i,\ i = 1, 4$ (8 flops).

A short reminder about the Roofline Model.

- SIMD:

AVX instructions:

In one clock cycle, do:

$y_i = a_i\, x_i + b_i,\ i = 1, 4$ (8 flops).

So, suppose you have a:

2×8 core machine at 2.6 10^9 cycles/second (Sandy Bridge).

The peak performance is:

2×8

A short reminder about the Roofline Model.

- SIMD:

AVX instructions:

In one clock cycle, do:

$y_i = a_i\, x_i + b_i,\ i = 1, 4$ (8 flops).

So, suppose you have a:

2×8 core machine at $2.6\ 10^9$ cycles/second (Sandy Bridge).

The peak performance is:

$2 \times 8 \times 2.6\ 10^9$

A short reminder about the Roofline Model.

- SIMD:

AVX instructions:

In one clock cycle, do:

$y_i = a_i\, x_i + b_i,\ i = 1, 4$ (8 flops).

So, suppose you have a:

2×8 core machine at $2.6\ 10^9$ cycles/second (Sandy Bridge).

The peak performance is:

$2 \times 8 \times 2.6\ 10^9 \times 8$

A short reminder about the Roofline Model.

- SIMD:

AVX instructions:

In one clock cycle, do:

$y_i = a_i\, x_i + b_i,\ i = 1, 4$ (8 flops).

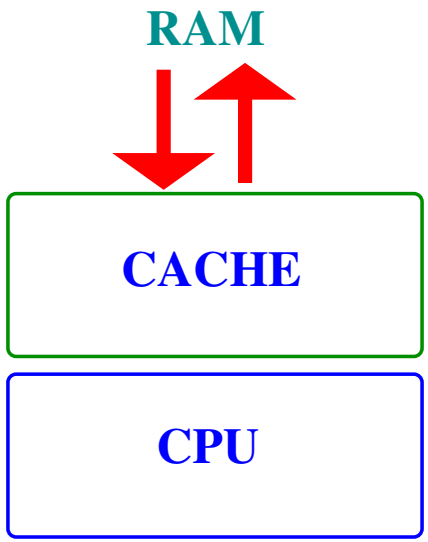
So, suppose you have a:

2×8 core machine at $2.6\ 10^9$ cycles/second (Sandy Bridge).

The peak performance is:

$2 \times 8 \times 2.6\ 10^9 \times 8 = 332$ Gflops/second.

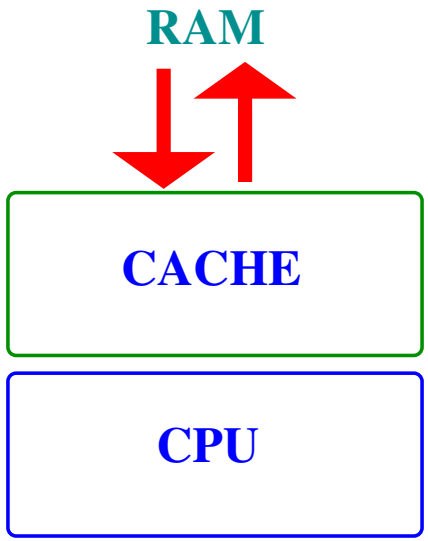
Arithmetic intensity and the Roofline Model



Actually, in most cases, performances are limited by the bandwidth of the machine.

Attainable GFlops/sec?

Arithmetic intensity and the Roofline Model



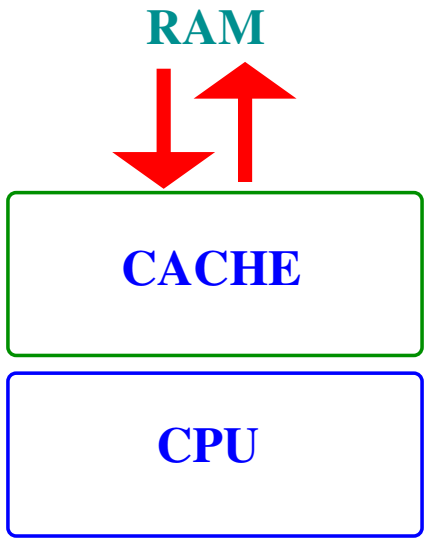
Actually, in most cases, performances are limited by the bandwidth of the machine.

Attainable GFlops/sec?

Arithmetic Intensity

I_a = number of operations / number of double words exchanged.

Arithmetic intensity and the Roofline Model



Actually, in most cases, performances are limited by the bandwidth of the machine.

Attainable GFlops/sec?

Arithmetic Intensity

I_a = number of operations / number of double words exchanged.

Attainable GFlops/sec =

min(Peak Performance, Peak Memory Bandwidth $\times$ I_a).

Arithmetic intensity and the Roofline Model

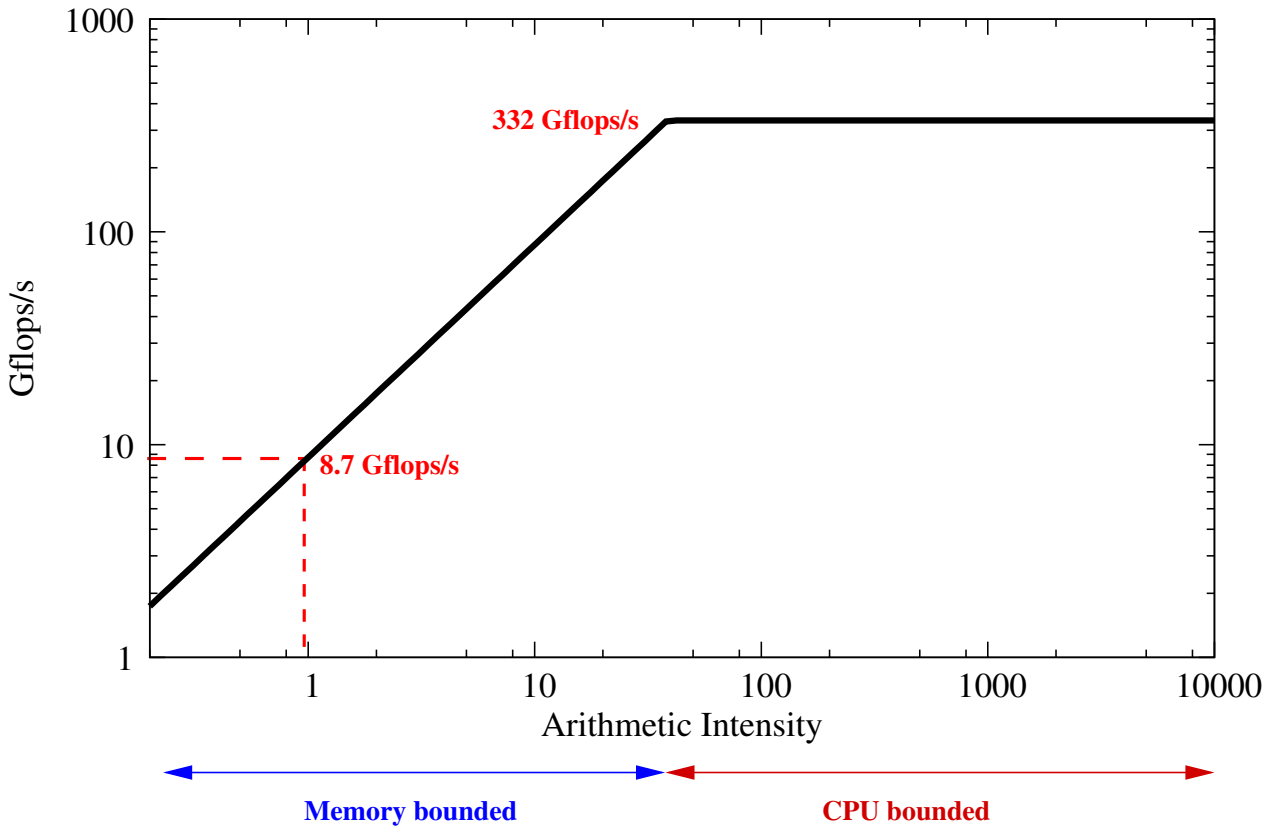
Sandy Bridge, 2×8 core: bandwidth $\simeq 8.7$ Giga-doubles/s.

Arithmetic intensity and the Roofline Model

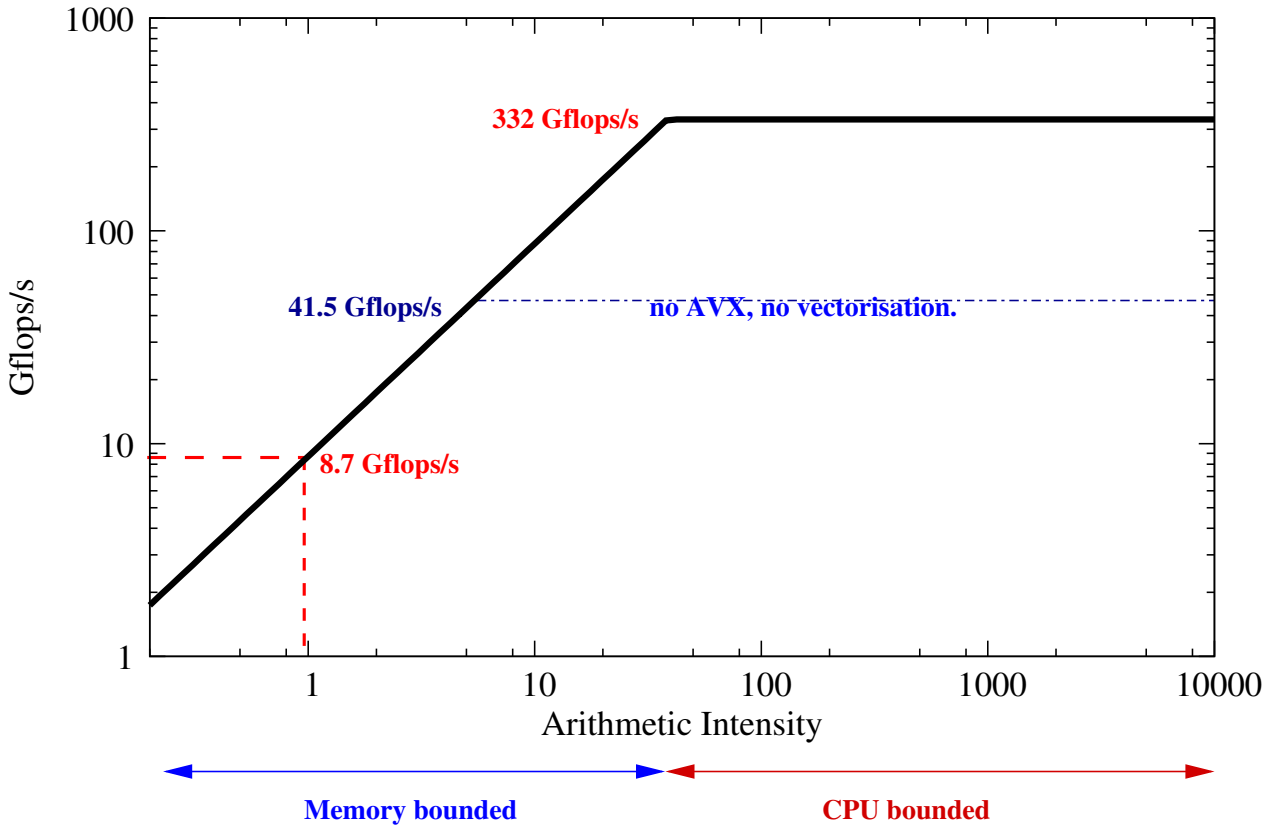
Sandy Bridge, 2×8 core: bandwidth $\simeq 8.7$ Giga-doubles/s.

To get the Peak Performance you need: $I_a \simeq 38.7$!

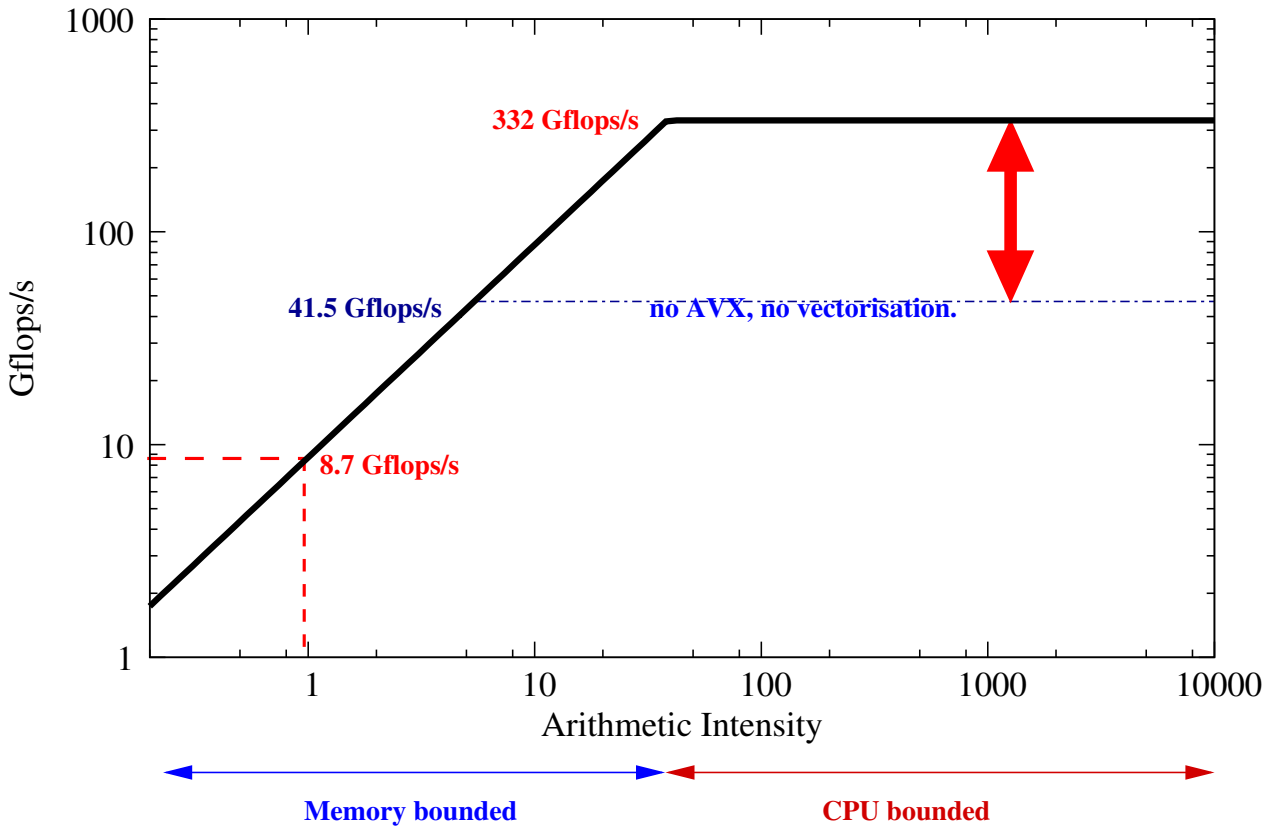
The Roofline Model



The Roofline Model



The Roofline Model



Williams S. et al: *Roofline: An Insightful Visual Performance Model for Multicore Architectures* – Commun. ACM, 1999.

Arithmetic intensity and the Roofline Model: some examples

Unit used: double.

1. Dot product $s = \sum_{i=1}^{n} x_i.y_i$: $I_a = 1$.

Arithmetic intensity and the Roofline Model: some examples

Unit used: double.

1. Dot product $s = \sum_{i=1}^{n} x_i.y_i$: $I_a = 1$.
2. Apply the 7 points Laplacian stencil in 3d: $I_a = 8/8 = 1$.

Arithmetic intensity and the Roofline Model: some examples

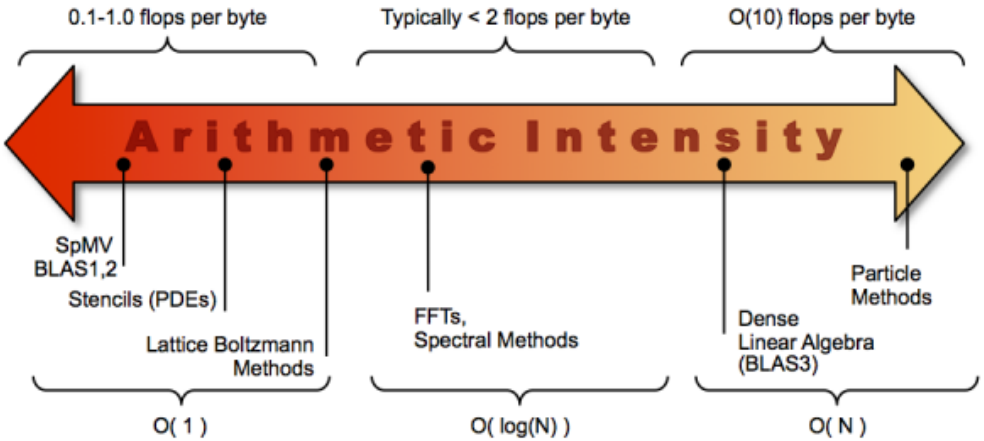
Unit used: double.

1. Dot product $s = \sum_{i=1}^{n} x_i.y_i$: $I_a = 1$.
2. Apply the 7 points Laplacian stencil in 3d: $I_a = 8/8 = 1$.
3. Matrix $\times$ Matrix product $C = A.B$: $I_a = 2\ n^3/4\ n^2 = \mathcal{O}(n)$.

Arithmetic intensity and the Roofline Model: some examples

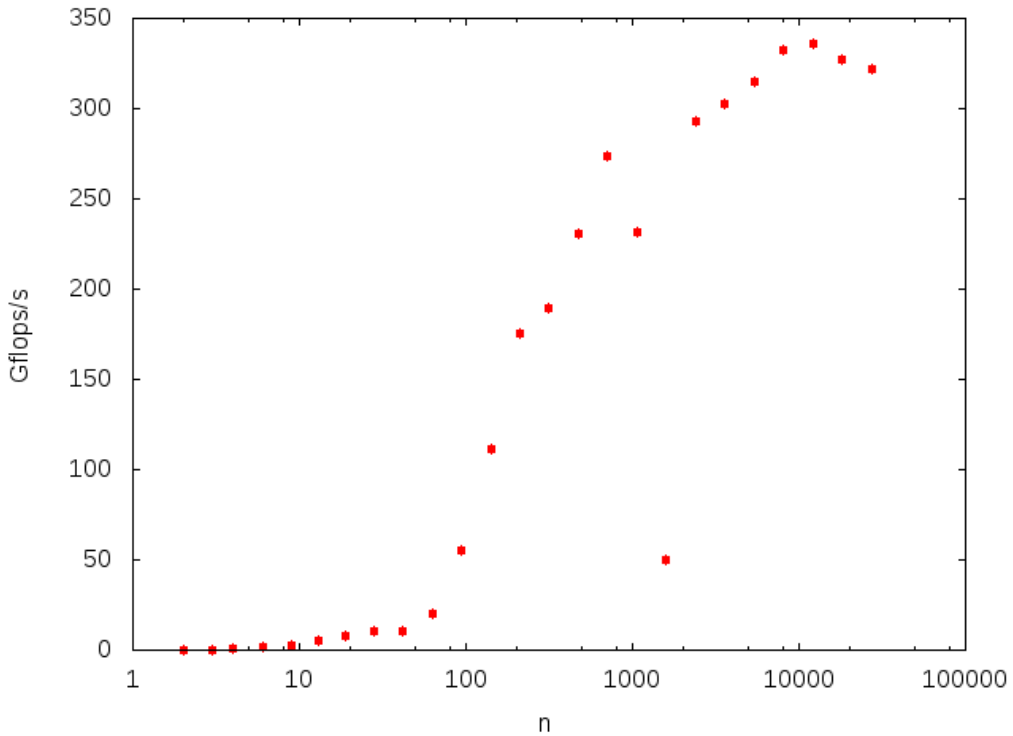
Unit used: double.

1. Dot product $s = \sum_{i=1}^{n} x_i.y_i$: $I_a = 1$.
2. Apply the 7 points Laplacian stencil in 3d: $I_a = 8/8 = 1$.
3. Matrix $\times$ Matrix product $C = A.B$: $I_a = 2\ n^3/4\ n^2 = \mathcal{O}(n)$.



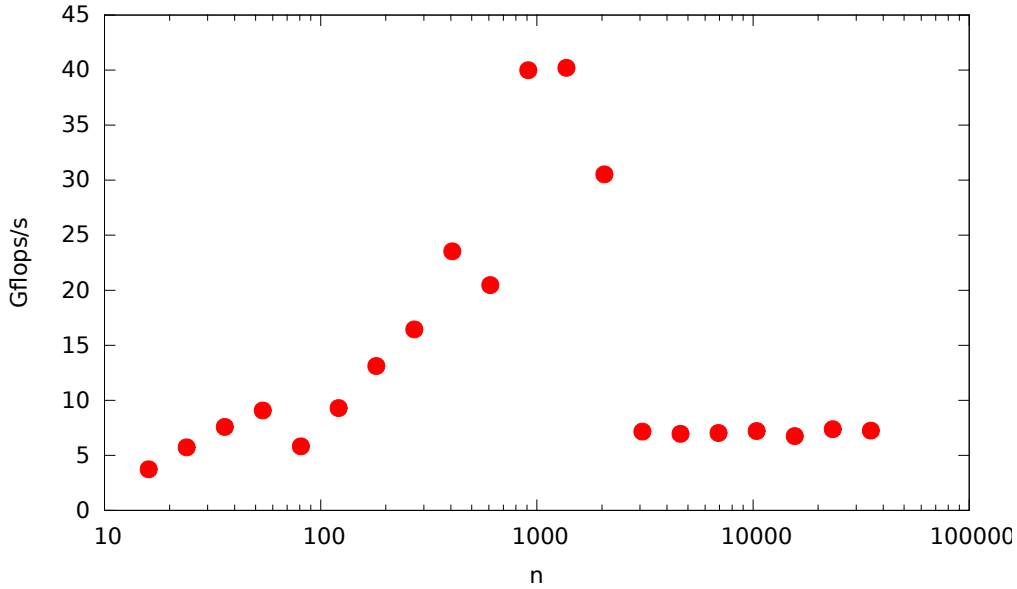
Arithmetic intensity: experiences

Matrix $\times$ Matrix product (DGEMM, Intel mkl parallel version):



Arithmetic intensity: experiences

Matrix $\times$ Vector product (DGEMV, Intel mkl parallel version):

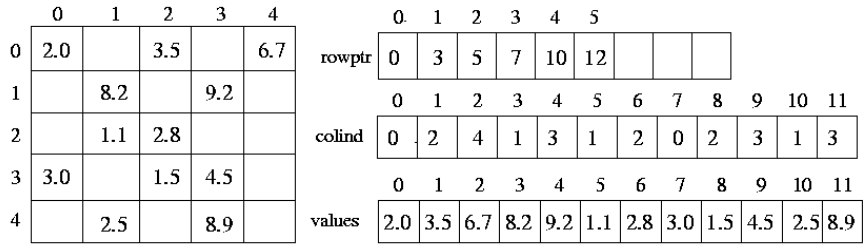


Arithmetic intensity and the Roofline Model: some examples

Apply 3-d 7 point Laplacian stencil, as a matrix stored in CSR/CSL format (ie. store coefficients and row indices):

Arithmetic intensity and the Roofline Model: some examples

Apply 3-d 7 point Laplacian stencil, as a matrix stored in CSR/CSL format (ie. store coefficients and row indices):



Arithmetic intensity and the Roofline Model: some examples

Apply 3-d 7 point Laplacian stencil, as a matrix stored in CSR/CSL format (ie. store coefficients and row indices):

	0	1	2	3	4
0	2.0		3.5		6.7
1		8.2		9.2	
2		1.1	2.8		
3	3.0		1.5	4.5	
4		2.5		8.9	

	0	1	2	3	4	5
rowptr	0	3	5	7	10	12

	0	1	2	3	4	5	6	7	8	9	10	11
colind	0	2	4	1	3	1	2	0	2	3	1	3

	0	1	2	3	4	5	6	7	8	9	10	11
values	2.0	3.5	6.7	8.2	9.2	1.1	2.8	3.0	1.5	4.5	2.5	8.9

- Algorithm Bdwth: 37/2 double; Flops: 13 $=>$ $I_a \simeq 0.7$.

Arithmetic intensity and the Roofline Model: some examples

Apply 3-d 7 point Laplacian stencil, as a matrix stored in CSR/CSL format (ie. store coefficients and row indices):

	0	1	2	3	4
0	2.0		3.5		6.7
1		8.2		9.2	
2		1.1	2.8		
3	3.0		1.5	4.5	
4		2.5		8.9	

	0	1	2	3	4	5
rowptr	0	3	5	7	10	12

	0	1	2	3	4	5	6	7	8	9	10	11
colind	0	2	4	1	3	1	2	0	2	3	1	3

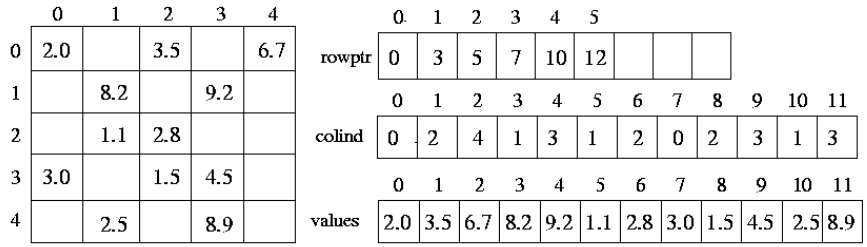
	0	1	2	3	4	5	6	7	8	9	10	11
values	2.0	3.5	6.7	8.2	9.2	1.1	2.8	3.0	1.5	4.5	2.5	8.9

- Algorithm Bdwth: 37/2 double; Flops: 13 $\Rightarrow$ $I_a \simeq 0.7$.
- Machine Bdwth: 8.73 Giga doubles/s

  $\Rightarrow$ Attainable $= 0.7 \times 8.73 = 6.11$ Gflops.

Arithmetic intensity and the Roofline Model: some examples

Apply 3-d 7 point Laplacian stencil, as a matrix stored in CSR/CSL format (ie. store coefficients and row indices):



	0	1	2	3	4
0	2.0		3.5		6.7
1		8.2		9.2	
2		1.1	2.8		
3	3.0		1.5	4.5	
4		2.5		8.9	

rowptr	0	1	2	3	4	5
	0	3	5	7	10	12

colind	0	1	2	3	4	5	6	7	8	9	10	11
	0	2	4	1	3	1	2	0	2	3	1	3

values	0	1	2	3	4	5	6	7	8	9	10	11
	2.0	3.5	6.7	8.2	9.2	1.1	2.8	3.0	1.5	4.5	2.5	8.9

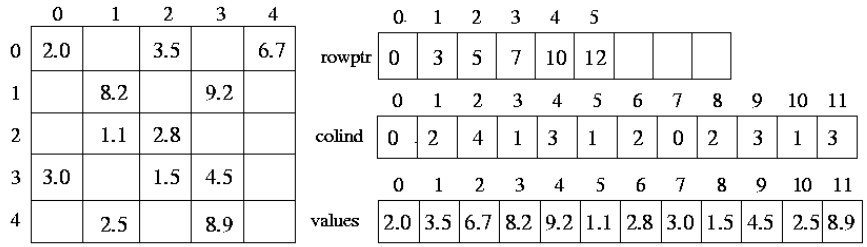
- Algorithm Bdwth: 37/2 double; Flops: 13 $=>$ $I_a \simeq 0.7$.
- Machine Bdwth: 8.73 Giga doubles/s

$=>$ Attainable $= 0.7 \times 8.73 = 6.11$ Gflops.

Measured: 6.42 Gflops.

Arithmetic intensity and the Roofline Model: some examples

Apply 3-d 7 point Laplacian stencil, as a matrix stored in CSR/CSL format (ie. store coefficients and row indices):



	0	1	2	3	4
0	2.0		3.5		6.7
1		8.2		9.2	
2		1.1	2.8		
3	3.0		1.5	4.5	
4		2.5		8.9	

	0	1	2	3	4	5
rowptr	0	3	5	7	10	12

	0	1	2	3	4	5	6	7	8	9	10	11
colind	0	2	4	1	3	1	2	0	2	3	1	3

	0	1	2	3	4	5	6	7	8	9	10	11
values	2.0	3.5	6.7	8.2	9.2	1.1	2.8	3.0	1.5	4.5	2.5	8.9

- Algorithm Bdwth: 37/2 double; Flops: 13 $\Rightarrow$ $I_a \simeq 0.7$.
- Machine Bdwth: 8.73 Giga doubles/s

$\Rightarrow$ Attainable $= 0.7 \times 8.73 = 6.11$ Gflops.

Measured: 6.42 Gflops.

Note: bounded to $1.15 \times 8.73 \simeq 10$ Gflops/s whatever the data structure.

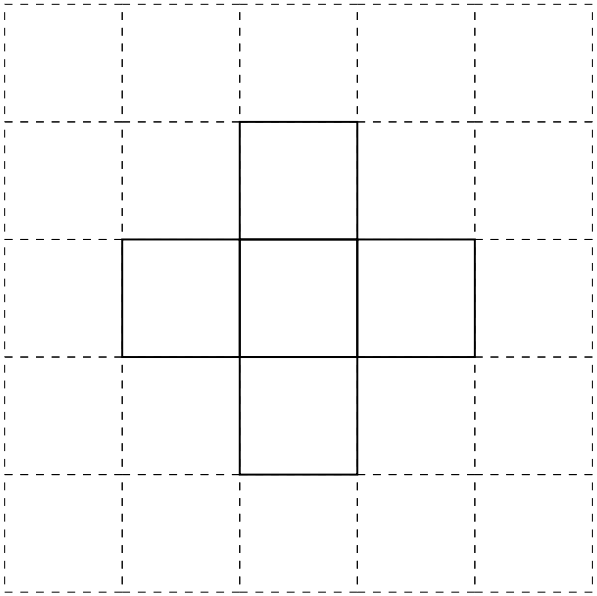
No hope?

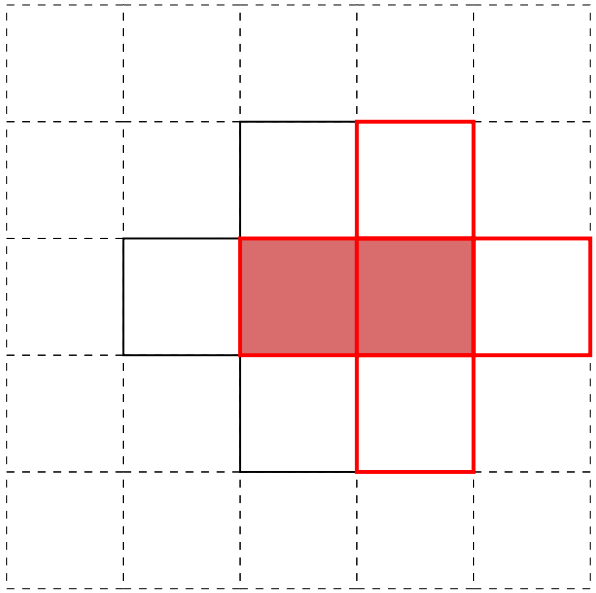
Is the 7-points stencil ($I_a = 1$) bounded to 8.73 Gflop/s?

No hope?

Is the 7-points stencil ($I_a = 1$) bounded to 8.73 Gflop/s?

No! We can re-use data.





$$Y = \mathrm{Stencil}_7(\mathrm{U}) + \sum_{\mathrm{i}=0}^{\mathrm{k}} \alpha_{\mathrm{i}} \mathrm{V}_{\mathrm{i}}.$$

k	$Max.I_a$	Gflops/s
0	4.0	34.8
1	3.3	29.0
2	3.0	26.1
3	2.8	24.4

Avoid:

- dot products,
- sparse matrices (incomplete LU preconditioning),
- linear combination of large vectors.
- methods with a limited parallelism.
- ... low intensity methods.

Avoid:

- dot products,
- sparse matrices (incomplete LU preconditioning),
- linear combination of large vectors.
- methods with a limited parallelism.
- ... low intensity methods.

Prefer:

- ... high intensity methods,
- embarrassingly parallel methods.

Runge–Kutta methods and linear problems.

$$\frac{du}{dt} = F(t,u), \quad u(t_0) = u_0,$$

$$F \ : \ [t_0, +\infty[\times \mathbb{R}^n \to \mathbb{R}^n.$$

Runge–Kutta methods and linear problems.

$$\frac{du}{dt} = F(t,u), \quad u(t_0) = u_0,$$

$$F \ : \ [t_0, +\infty[\times \mathbb{R}^n \to \mathbb{R}^n.$$

Runge–Kutta method, with s stages:

$$\begin{aligned} k_i &= F\left(x_0 + c_i \delta t, y_0 + \delta t (\textstyle\sum_{j=1}^{s} a_{ij} k_j)\right) \quad j = 1, \ldots, s, \\ u_1 &= u_0 + \delta t \sum_{j=1}^{s} b_j k_j. \end{aligned}$$

Runge–Kutta methods

$$
\begin{array}{c|cccc}
c_1 & a_{11} & a_{12} & \cdots & a_{1s-1} & a_{1s} \\
c_2 & a_{21} & a_{22} & \cdots & a_{2s-1} & a_{2s} \\
\vdots & \vdots & & \ddots & & \vdots \\
c_s & a_{s1} & a_{s2} & \cdots & a_{ss-1} & a_{ss} \\
\hline
 & b_1 & b_2 & \cdots & b_{s-1} & b_s
\end{array}
$$

Butcher array.

Runge–Kutta methods

$$
\begin{array}{c|cccc}
c_1 & a_{11} & a_{12} & \cdots & a_{1s-1} & a_{1s} \\
c_2 & a_{21} & a_{22} & \cdots & a_{2s-1} & a_{2s} \\
\vdots & \vdots & & \ddots & & \vdots \\
c_s & a_{s1} & a_{s2} & \cdots & a_{ss-1} & a_{ss} \\
\hline
 & b_1 & b_2 & \cdots & b_{s-1} & b_s
\end{array}
$$

Butcher array.

Classification:

- Implicit method: there exist $a_{ij} \neq 0$ for $j \geq 0$. Solve a system of size $s \times n$. Typical example: Radau5.

Runge–Kutta methods

$$
\begin{array}{c|cccc}
c_1 & a_{11} & a_{12} & \cdots & a_{1s-1} & a_{1s} \\
c_2 & a_{21} & a_{22} & \cdots & a_{2s-1} & a_{2s} \\
\vdots & \vdots & & \ddots & & \vdots \\
c_s & a_{s1} & a_{s2} & \cdots & a_{ss-1} & a_{ss} \\
\hline
 & b_1 & b_2 & \cdots & b_{s-1} & b_s
\end{array}
$$

Butcher array.

Classification:

- Implicit method: there exist $a_{ij} \neq 0$ for $j \geq 0$. Solve a system of size $s \times n$. Typical example: Radau5.
- Diagonally Implicit: $\forall i, \forall j > i,\ a_{ij} = 0$. Solve s independent systems of size n; (generally: $\forall i\ a_{ii} = \gamma$).

Runge–Kutta methods

c_1	a_{11}	a_{12}	$\cdots$	a_{1s-1}	a_{1s}
c_2	a_{21}	a_{22}	$\cdots$	a_{2s-1}	a_{2s}
$\vdots$	$\vdots$		$\ddots$		$\vdots$
c_s	a_{s1}	a_{s2}	$\cdots$	a_{ss-1}	a_{ss}
	b_1	b_2	$\cdots$	b_{s-1}	b_s

Butcher array.

Classification:

- Implicit method: there exist $a_{ij} \neq 0$ for $j \geq 0$. Solve a system of size $s \times n$. Typical example: Radau5.
- Diagonally Implicit: $\forall i, \forall j > i,\ a_{ij} = 0$. Solve s independent systems of size n; (generally: $\forall i\ a_{ii} = \gamma$).
- Explicit: $\forall i, \forall j \geq i,\ a_{ij} = 0$. No system!

Runge–Kutta methods: linear problems

Scalar equation:

$$\frac{du}{dt} = \lambda u, \quad \lambda \in \mathbb{C}$$

Runge–Kutta methods: linear problems

Scalar equation:

$$\frac{du}{dt} = \lambda u, \quad \lambda \in \mathbb{C}$$

Set: $z = \lambda \delta t$. Then:

- if the method is (diagonally) implicit: $u_1 = Q(z)u_0$ where Q is a rational fraction.
- if the method is explicit: $u_1 = Q(z)u_0$, but Q is a polynomial.

Runge–Kutta methods: stability

$$\mathcal{D} = \{z \in \mathbb{C},\ |Q(z)| < 1\}.$$

Runge–Kutta methods: stability

$$\mathcal{D} = \{z \in \mathbb{C},\ |Q(z)| < 1\}.$$

Important:

- Explicit method: $\mathcal{D}$ is bounded in all directions.
  (size of $\mathcal{D}$ is about 1).
- Implicit method: $\mathcal{D}$ can be unbounded in some directions:
  - A-stability: $\mathbb{C}^- \subset \mathcal{D}$.
  - L-stability: A-stability and $Q(z) \to 0$ when $\mathrm{Re(z)} \to -\infty$.

Runge–Kutta methods: stability

Recall that for:

$$\frac{du}{dt} = \varepsilon \Delta u,$$

we have $z \simeq \varepsilon \delta t / h^2$. Then:

- *Classical* explicit methods: $dt < Ch^2/\varepsilon$,

Runge–Kutta methods: stability

Recall that for:

$$\frac{du}{dt} = \varepsilon \Delta u,$$

we have $z \simeq \varepsilon \delta t / h^2$. Then:

- *Classical* explicit methods: $dt < Ch^2/\varepsilon$,
- (Diagonally) implicit A-stable methods: no bound to dt (except precision!)

Runge–Kutta methods: stability

Recall that for:

$$\frac{du}{dt} = \varepsilon \Delta u,$$

we have $z \simeq \varepsilon \delta t / h^2$. Then:

- *Classical* explicit methods: $dt < Ch^2/\varepsilon$,
- (Diagonally) implicit A-stable methods: no bound to dt (except precision!)

Is there some room between these methods?

Runge–Kutta methods: stability

Recall that for:

$$\frac{du}{dt} = \varepsilon \Delta u,$$

we have $z \simeq \varepsilon \delta t / h^2$. Then:

- *Classical* explicit methods: $dt < Ch^2/\varepsilon$,
- (Diagonally) implicit A-stable methods: no bound to dt (except precision!)

Is there some room between these methods?

- Fully implicit methods (Radau5: 3 steps, order 5) limited to relatively small systems.
- Heat equation: use diagonally implicit methods and thus solve s linear systems. You learned it at school! I claim it is not so true.

Explicit Runge–Kutta methods with improved stability domain

An explicit method of order p as at least p stages. Example: the classical RK4 method:

$$Q(z) = 1 + z + z^2/2 + z^3/6 + z^4/24.$$

Explicit Runge–Kutta methods with improved stability domain

An explicit method of order p as at least p stages. Example: the classical RK4 method:

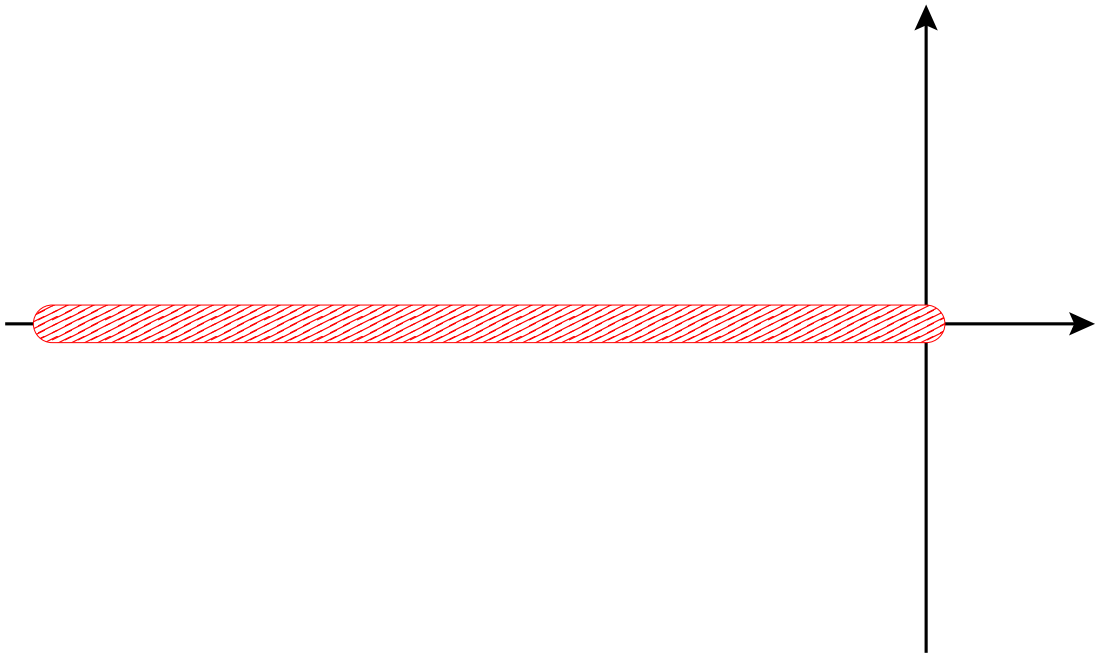
$$Q(z) = 1 + z + z^2/2 + z^3/6 + z^4/24.$$

Idea: for order p, use $s > p$ stages:

$$Q(z) = \sum_{i=0}^{p} \frac{x^p}{p!} + \sum_{i=p+1}^{s} a_i x^i,$$

and choose $a_i, i = p+1, s$ so that $\mathcal{D}$ as the largest possible extension along the real negative axis.

Explicit Runge–Kutta methods with improved stability domain



The stability domain $\mathcal{D}$ contains a narrow strip along the real negative axis.

Explicit Runge–Kutta methods with improved stability domain

- 1rst generation of methods. Example: Dumka (Lebedev).
  Optimal polynomial in the form

$$\prod_{i=0}^{s}(1-\alpha_i z).$$

  When solving $du/dt = Au$:

$$u_1 = \prod_{i=0}^{s}(I-\alpha_i \delta t A)u_0.$$

  Product of explicit Euler methods => stability problem, solved by a well chosen permutation of the a_i.
  A bit too tricky!

Explicit Runge–Kutta methods with improved stability domain

- 2nd generation: Rock methods (A. Abdulle). Quasi optimal polynomial.
  - for a given s, we have a set of $s - p$ orthogonal polynomials for a weight w.
  - the weight w is a polynomial.

Explicit Runge–Kutta methods with improved stability domain

- 2nd generation: Rock methods (A. Abdulle). Quasi optimal polynomial.
  - for a given s, we have a set of $s - p$ orthogonal polynomials for a weight w.
  - the weight w is a polynomial.

  Most difficult mathematical problems:
  - return from polynomials to RK. method (the Butcher array). Actually is is the composition of 2 RK. explicit methods.
  - proof of order.

  (use B-series, Butcher group).

Explicit Runge–Kutta methods with improved stability domain

- 2nd generation: Rock methods (A. Abdulle). Quasi optimal polynomial.
  - for a given s, we have a set of $s - p$ orthogonal polynomials for a weight w.
  - the weight w is a polynomial.

  Most difficult mathematical problems:
  - return from polynomials to RK. method (the Butcher array). Actually is is the composition of 2 RK. explicit methods.
  - proof of order.

  (use B-series, Butcher group).

Exists for order 2 and 4 (Rock2 and Rock4).
Domain extension along $\mathbb{R}^-$ grows like p^2 (proven for RK2, experimentally verified for RK4).

Explicit Runge–Kutta methods with improved stability domain

In the linear case:

1. First method: 3 terms recurrence formula.
2. Second method: a classical explicit RK. method

Explicit Runge–Kutta methods with improved stability domain

In the linear case:

1. First method: 3 terms recurrence formula.
2. Second method: a classical explicit RK. method $\rightarrow$ polynomial

Explicit Runge–Kutta methods with improved stability domain

In the linear case:

1. First method: 3 terms recurrence formula.
2. Second method: a classical explicit RK. method $\rightarrow$ polynomial $\rightarrow$ use Horner scheme to evaluate the polynomial.

Explicit Runge–Kutta methods with improved stability domain

In the linear case:

1. First method: 3 terms recurrence formula.
2. Second method: a classical explicit RK. method $\rightarrow$ polynomial $\rightarrow$ use Horner scheme to evaluate the polynomial.

For $du/dt = Au$, we only need to compute:

- $v \leftarrow \alpha Ax + \beta y$,
- $v \leftarrow \alpha Ax + \beta y + \gamma z$.

Explicit Runge–Kutta methods with improved stability domain

In the linear case:

1. First method: 3 terms recurrence formula.
2. Second method: a classical explicit RK. method $\rightarrow$ polynomial $\rightarrow$ use Horner scheme to evaluate the polynomial.

For $du/dt = Au$, we only need to compute:

- $v \leftarrow \alpha Ax + \beta y$,
- $v \leftarrow \alpha Ax + \beta y + \gamma z$.

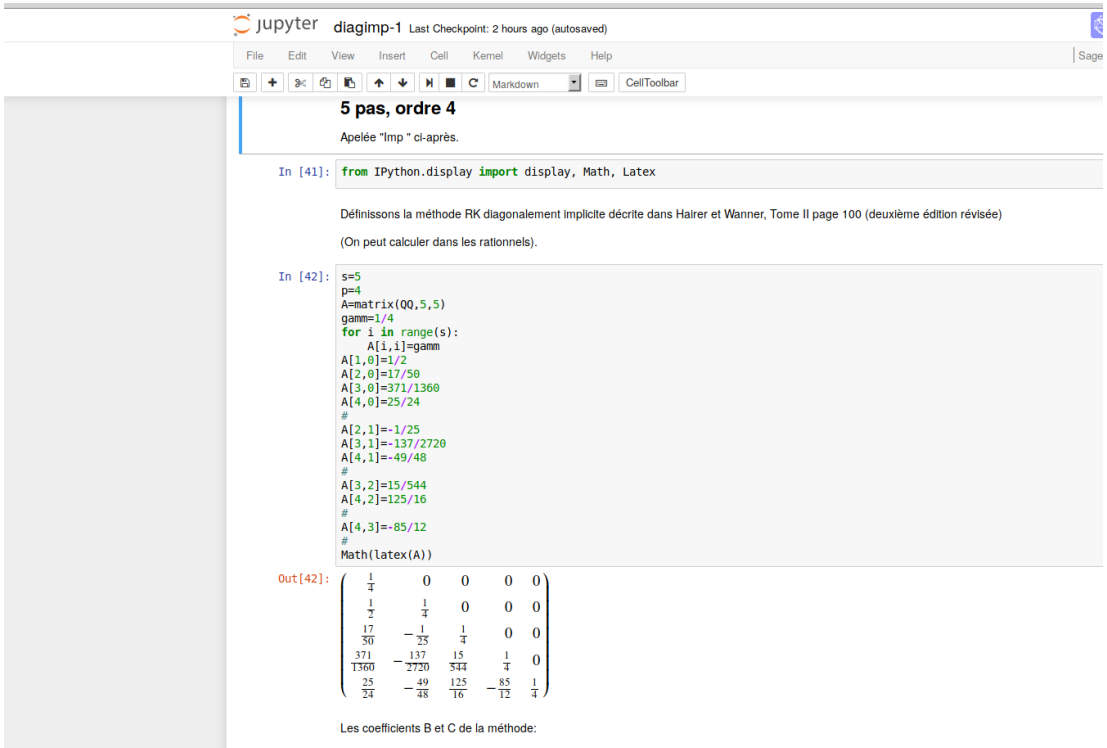
Note: for large parabolic problems, *never* use A. Abdulle code: very good, but does a lot of copy.

Rock methods or diagonally implicit methods: which are the best?

Compare Rock2 with 4 order diagonally implicit methods, Rock2 with 2 order ones, in dimension 1, classical finite difference approximation.

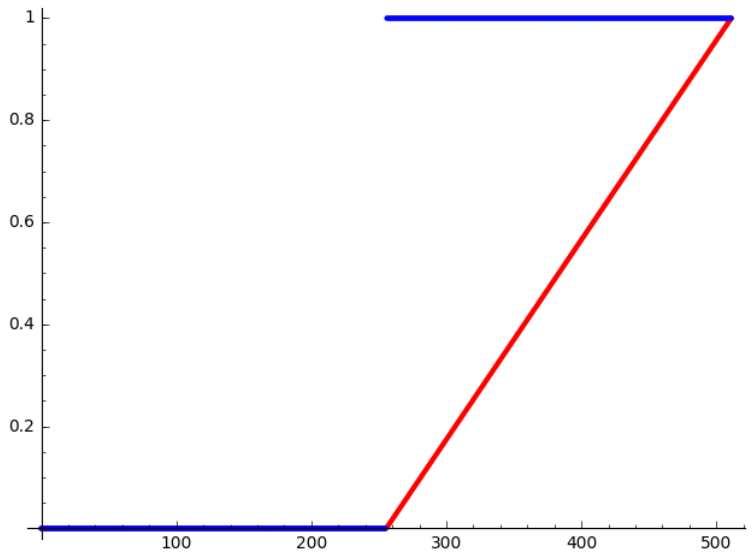
Rock methods or diagonally implicit methods: which are the best?

Compare Rock2 with 4 order diagonally implicit methods, Rock2 with 2 order ones, in dimension 1, classical finite difference approximation.
Apply polynomial or rational fraction to eigenvalues of the stencil.
Thanks to Sage :-)



Rock methods or diagonally implicit methods: which are the best?

Initial conditions:



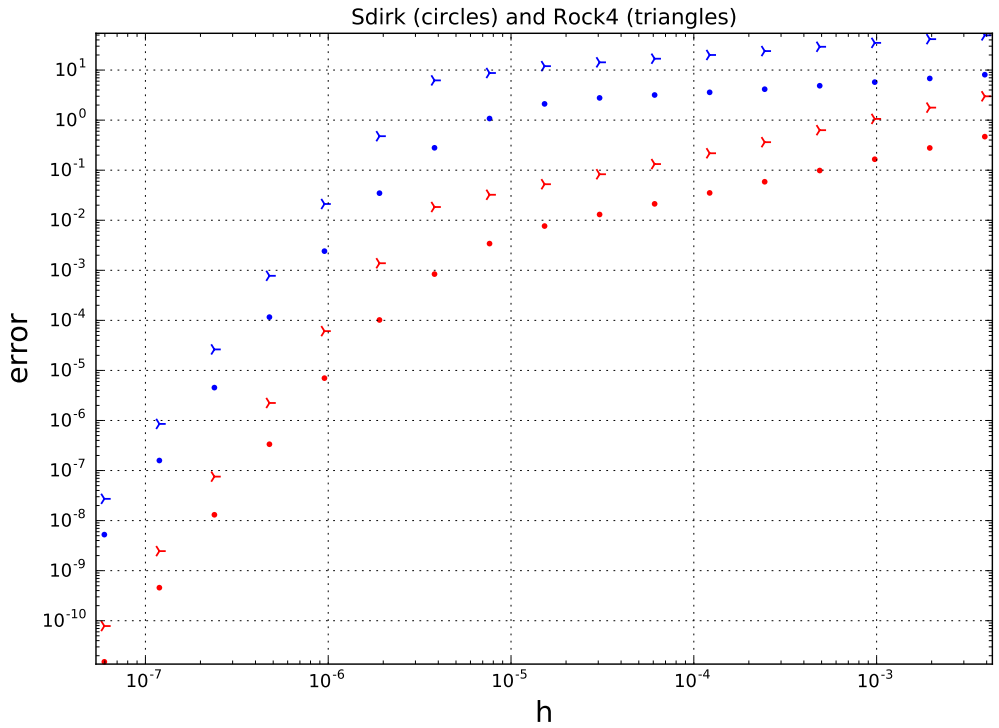
regular(red), irregular (blue).

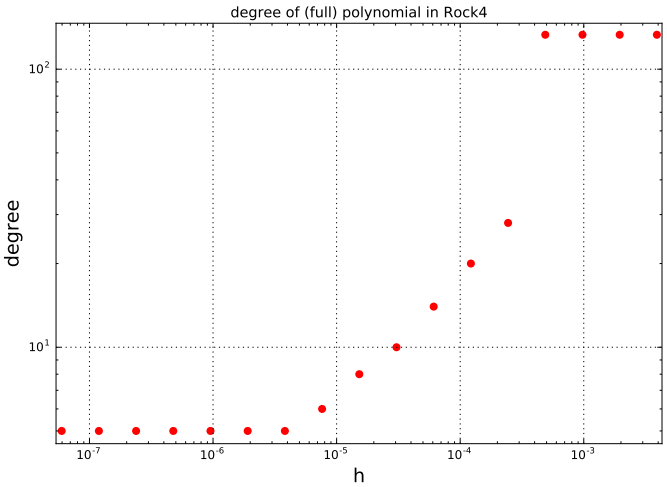
Rock methods or diagonally implicit methods: which are the best?

$$\begin{pmatrix} 1/4 & 0 & 0 & 0 & 0 \\ 1/2 & 1/4 & 0 & 0 & 0 \\ 17/50 & -1/25 & 1/4 & 0 & 0 \\ 371/1360 & -137/2720 & 15/544 & 1/4 & 0 \\ 25/24 & -49/48 & 125/16 & -85/12 & 1/4 \end{pmatrix}$$

A method of order 4. See Hairer–Wanner T.II page 100.

Rock methods or diagonally implicit methods: which are the best?





- Regular: Error $\simeq 10^{-5}$. $h_{\mathrm{Sdirk}} = 10^{-6}$, $h_{\mathrm{Rock4}} = 6\ 10^{-7}$.
- Irregular: Error $\simeq 10^{-5}$. $h_{\mathrm{Sdirk}} = 4\ 10^{-7}$, $h_{\mathrm{Rock4}} = 2\ 10^{-7}$.

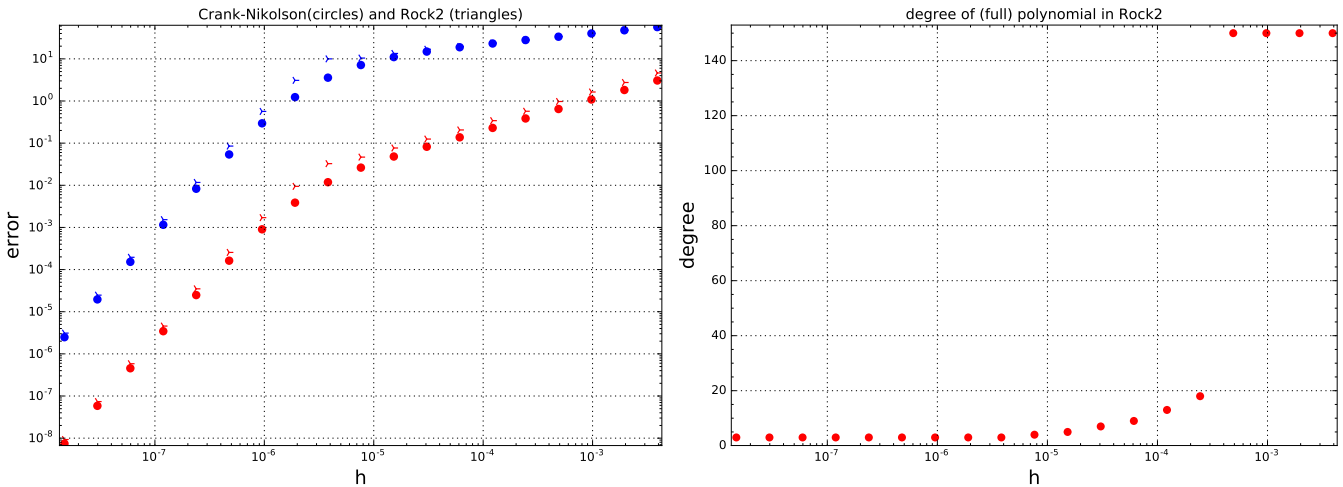
Rock4: compare solving 5 linear systems with less than 10 matrix-vector products (degree of polynomial 5).

Rock methods or diagonally implicit methods: which are the best?

Compare Crank-Nikolson and Rock2.

$$R = \frac{1 + z/2}{1 - z/2}$$

(A-stable but not L-stable!).



Quite the same precision (both cases). Can you solve a system at the cost of 5 matrix-vector products?

An experience with Rock4 and the 7-points Laplacian stencil (3d)

An experience with Rock4 and the 7-points Laplacian stencil (3d)

- Organize loops so as to maximize data reuse.
- Use a Horner scheme for the second part (W) of Rock4 formula (and say goodbye to error estimation).

$\simeq 25$ Gflop/s.

An experience with Rock4 and the 7-points Laplacian stencil (3d)

- Organize loops so as to maximize data reuse.
- Use a Horner scheme for the second part (W) of Rock4 formula (and say goodbye to error estimation).

$\simeq$ 25 Gflop/s.

Can we do better?

We must improve the arithmetic intensity of... the stencil.

Improve stencil arithmetic intensity

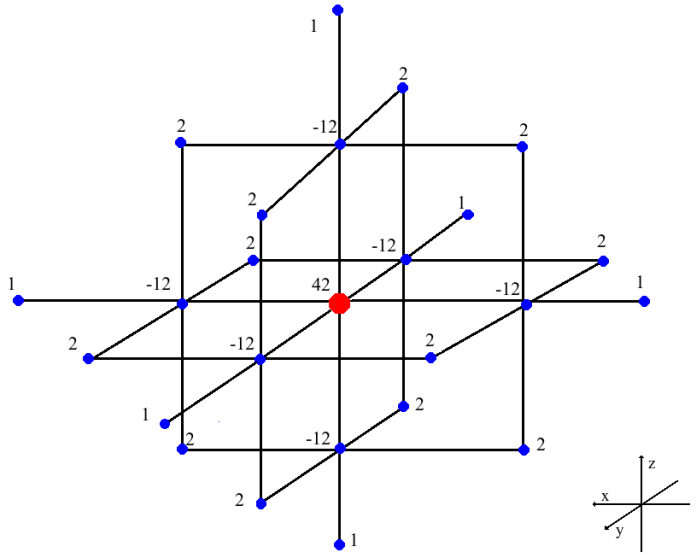
Use Δ^2 stencil.

- Horner scheme by pairs,
- Use Dumka method, grouping roots by pairs.

Improve stencil arithmetic intensity

Use Δ^2 stencil.

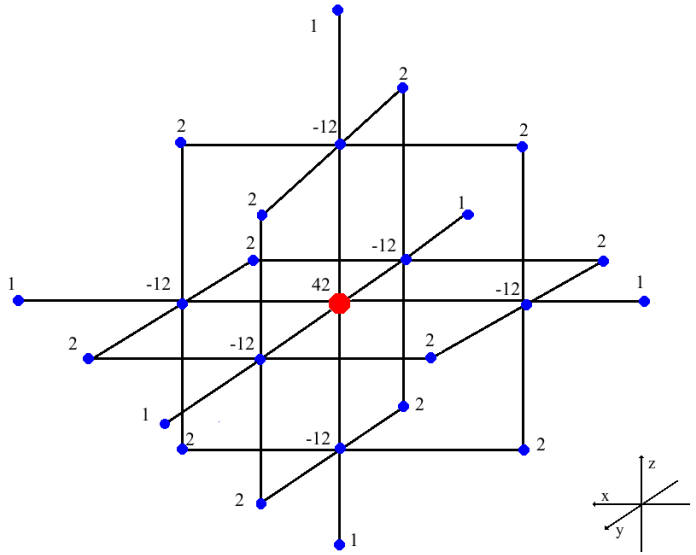
- Horner scheme by pairs,
- Use Dumka method, grouping roots by pairs.

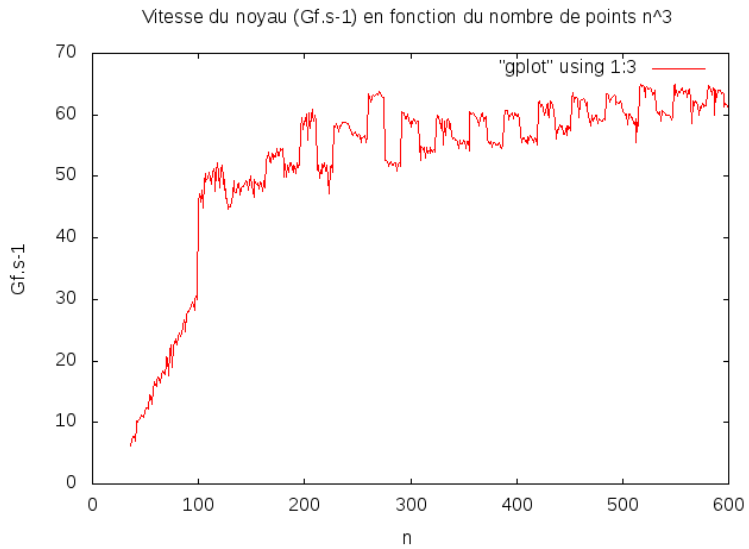


Improve stencil arithmetic intensity

Use Δ^2 stencil.

- Horner scheme by pairs,
- Use Dumka method, grouping roots by pairs.





Stability?

Can we do better?

High intensity stencils with Discontinuous Galerkin methods

High intensity stencils with Discontinuous Galerkin methods

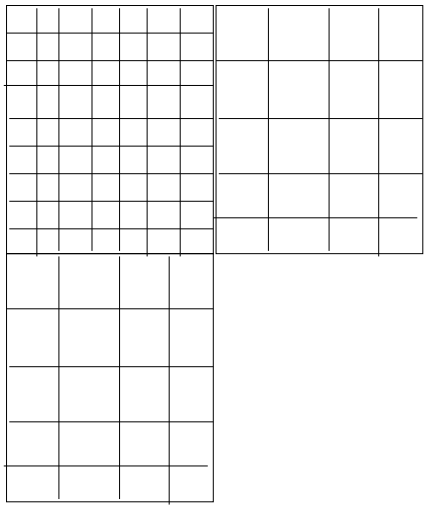
Advantages of DG:

- In a 3d Cartesian mesh: we get a 7 *matrices stencil.*
- High intensity I_a.
- Free choice of the polynomial basis.

High intensity stencils with Discontinuous Galerkin methods

Advantages of DG:

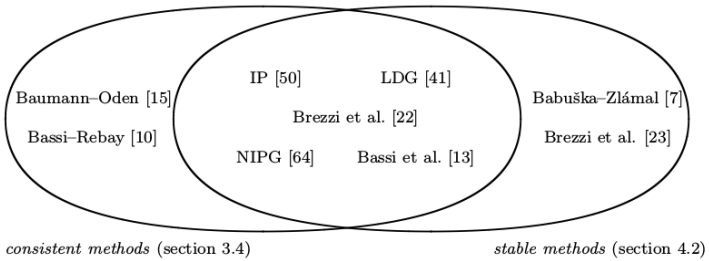
- In a 3d Cartesian mesh: we get a 7 *matrices stencil.*
- High intensity I_a.
- Free choice of the polynomial basis.



(quite easy with DG!)

High intensity stencils with Discontinuous Galerkin methods

Which DG method?



Best choice seems to be the Interior Penalty (IP) method.

Arnold, D. et al.: *Unified analysis of discontinuous Galerkin methods for elliptic problems* – SIAM J. Numer. Anal. 2001/02.

DG / Ip Method

Unknowns are σ_h and u_h.

$$\int_K \sigma_h.\tau dx = -\int_K u_h \nabla.\tau dx + \int_{\partial K} \hat{u}_K \eta_K.\tau ds \quad \forall \tau \in \Sigma(K),$$
$$\int_K \sigma_h.\nabla v dw = \int_K f v dx + \int_{\partial K} \hat{\sigma}_K.\eta_K v ds \quad \forall v \in P(K).$$

Where $\hat{\sigma}_K$ and $\hat{u}_K$ are numerical fluxes.

DG / Ip Method

Unknowns are σ_h and u_h.

$$\begin{aligned}
\int_K \sigma_h.\tau dx = & -\int_K u_h \nabla.\tau dx + \int_{\partial K} \hat{u}_K \eta_K.\tau ds \quad \forall \tau \in \Sigma(K), \\
\int_K \sigma_h.\nabla v dw = & \int_K f v dx + \int_{\partial K} \hat{\sigma}_K.\eta_K v ds \quad \forall v \in P(K).
\end{aligned}$$

Where $\hat{\sigma}_K$ and $\hat{u}_K$ are numerical fluxes.

Let K_1 and K_2 be 2 neighbor elements with a common edge e.

$$\begin{aligned}
\phi(x) \in \mathbb{R}^d \quad : \quad \phi = \frac{1}{2}(\phi_1 + \phi_2) \qquad [\phi] = \phi_1.\eta_1 + \phi_2.\eta_2, \\
\phi(x) \in \mathbb{R} \quad : \quad \phi = \frac{1}{2}(\phi_1 + \phi_2) \qquad [\phi] = \phi_1\eta_1 + \phi_2\eta_2.
\end{aligned}$$

DG / Ip Method

Interior penalty method. Fluxes

$$\hat{u} = u_h, \quad \hat{\sigma} = \nabla_h u_h - \eta_e h_e^{-1}[u_h].$$

But one can eliminate σ_h:

Interior penalty method. Primal form

$$B_h(u_h, v) = \int_\Omega \nabla_h u_h . \nabla_h v dx - \int_\Gamma ([u_h].\nabla_h v + \nabla_h u_h.[v])ds + \int_\Gamma \alpha [u_h].[v] ds.$$

with $\alpha = \eta_e h_e^{-1}$ on each $e \in \mathcal{E}$.

Implement the method using:

- Legendre basis:

$$Q_{i,j,k} = P_{i,j,k}(x, y, z) = p_i(x)\ p_j(y)\ p_k(z),$$

  with:

$$p_l(s) = L_l((2s - h)/h),\ \ l = 0, \mathrm{degree}.$$

  (normalized to obtain an identity mass matrix).
- for degrees from 2 to 5 (thanks to SageMath software).

DG

Best results for degree 3:

- I_a grows with the degree of polynomials.
- Think to vectorization: nowadays computers like vectors of size divisible by 4.

DG: stencil for polynomials of degree 3

- $A_{i,i}$ is a 64×64 matrix with 4 non zero terms by line.
- If $i \neq j$, $A_{i,j}$ has 256 non zero terms, with a regular structure (loop on a 4×4 matrix to perform the product).

DG: stencil for polynomials of degree 3

- $A_{i,i}$ is a 64×64 matrix with 4 non zero terms by line.
- If $i \neq j$, $A_{i,j}$ has 256 non zero terms, with a regular structure (loop on a 4×4 matrix to perform the product).

I_a ?

- Flops:

$A_{i,j},\ i \neq j$	:	6	$\times$	512	$=$	3072	
$A_{i,i}$	:	1	$\times$	512	$=$	512	
Total	:					3584	flops.

DG: stencil for polynomials of degree 3

- $A_{i,i}$ is a 64×64 matrix with 4 non zero terms by line.
- If $i \neq j$, $A_{i,j}$ has 256 non zero terms, with a regular structure (loop on a 4×4 matrix to perform the product).

I_a ?

- Flops:

$A_{i,j},\ i \neq j$	:	6	$\times$	512	$=$	3072	
$A_{i,i}$	:	1	$\times$	512	$=$	512	
Total	:					3584	flops.

- Memory bandwidth: $8 \times 64 \ = 512$ (double).

So, $I_a = 7$ without any reuse of data.

DG: stencil for polynomials of degree 3

$I_a = 7$.

- Peak theoretical performance:
  $7 \times 8.73 = 61.2$ Gigaflops/second.

DG: stencil for polynomials of degree 3

$I_a = 7$.

- Peak theoretical performance:
  $7 \times 8.73 = 61.2$ Gigaflops/second.
- Measured (Rock4, W method using Horner scheme):
  $\partial_t u = \Delta U$: 67 Gigaflops/second.
  $\partial_t u = \Delta U + f(x)$: 66 Gigaflops/second.

Some data is reused.

The Poisson equation

Conjugate Gradient and Polynomial Preconditioning.

Chebyshev preconditioning:

Find $s \in \mathbb{P}_k$ which minimizes:

$$\max_{\lambda \in [a,b]} |1 - \lambda s(\lambda)|.$$

Solution is a shifted and scaled Chebyshev polynomial.

To solve $Ax = B$, use $M^{-1} = s(A)$ as preconditionner.

The Poisson equation

Conjugate Gradient and Polynomial Preconditioning.

Chebyshev preconditioning:

Find $s \in \mathbb{P}_k$ which minimizes:

$$\max_{\lambda \in [a,b]} |1 - \lambda s(\lambda)|.$$

Solution is a shifted and scaled Chebyshev polynomial.

To solve $Ax = B$, use $M^{-1} = s(A)$ as preconditionner.

Evaluation using the 3 terms recurrence formula.

See results of W. Vanroose:
http://calcul.math.cnrs.fr/IMG/pdf/poisson_vanroose.pdf – .

GCP

$$
\begin{aligned}
r_0: \quad & = b - Ax_0; u_0 = M^{-1} r_0; p_0 = u_0; \\
\text{for} \quad & i = 0, ... \text{ do :} \\
& s := Ap_i \\
& \alpha :=< r_i, u_i > / < s, p_i > \\
& x_{i+1} := x_i + \alpha p_i \\
& r_{i+1} := r_i - \alpha s \\
& u_{i+1} := M^{-1} r_{i+1} \\
& \beta :=< r_{i+1}, u_{i+1} > / < r_i, u_i > \\
& p_{i+1} := u_{i+1} + \beta p_i
\end{aligned}
$$

GCP

r_0: $= b - Ax_0; u_0 = M^{-1} r_0; p_0 = u_0;$

for $i = 0, \ldots.$ do :

$s := Ap_i$

$\alpha := < r_i, u_i > / < s, p_i >$

$r_{i+1} := r_i - \alpha s$

$u_{i+1} := M^{-1} r_{i+1}$

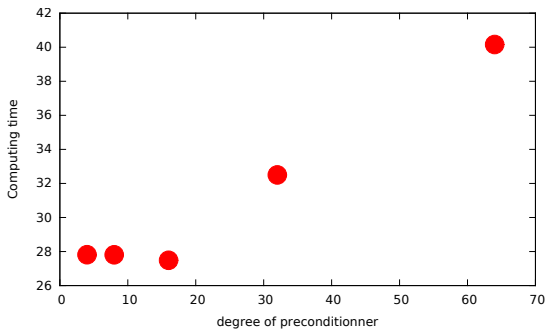
$\beta := < r_{i+1}, u_{i+1} > / < r_i, u_i >$

$x_{i+1} := x_i + \alpha p_i$

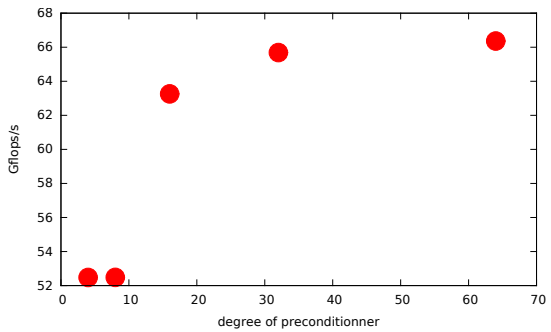
$p_{i+1} := u_{i+1} + \beta p_i$

GCP: results

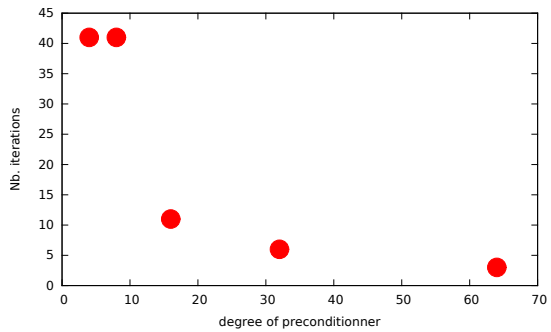
Grid 128^3 elements (512^3 unknowns), $-\Delta u = f$.



Computing time, best: degree $= 16$



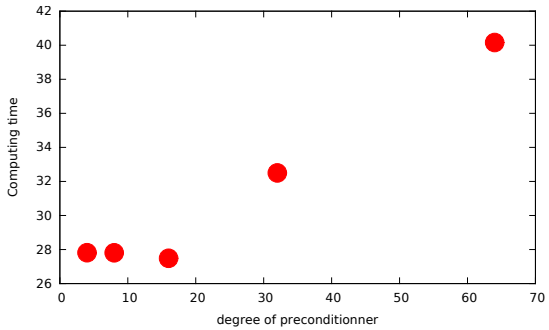
Gflops/s.



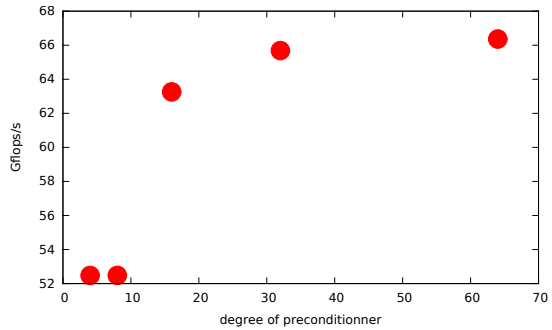
Nb. iterations

GCP: results

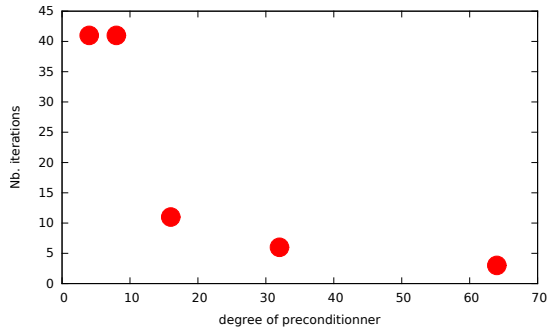
Grid 128^3 elements (512^3 unknowns), $-\Delta u = f$.



Computing time, best: degree $= 16$

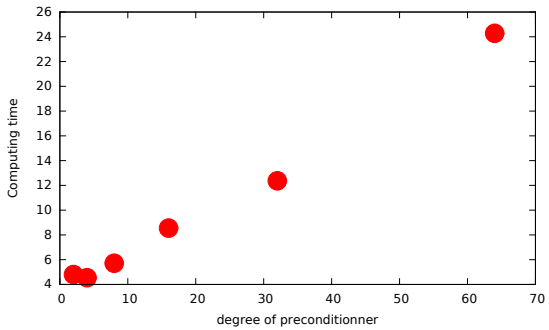


Gflops/s.

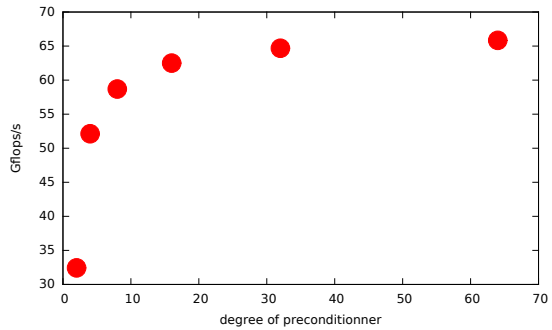


Nb. iterations

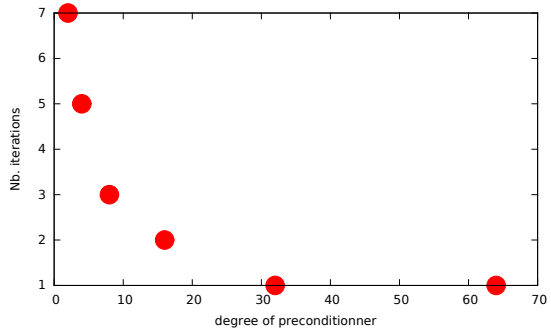
Grid 128^3 elements (512^3 unknowns), $-\Delta u + 0.01u = f$.



Computing time, best: degree $= 4$



Gflops/s.



Nb. iterations

Work in progress

Improve performances at “stencil” level. Not easy!

Work in progress

Improve performances at "stencil" level. Not easy!

Meet us at SMAI 2017!

Mini-Symposium EFFCALC: efficacité calculatoire des méthodes numériques.

L'évolution de l'architecture des machines (incluant tous les types de matériels et d'environnements, des CPU et GPU à l'Exascale), de leurs coeurs de calcul et de leur hiérarchie mémoire, nécessite de repenser les schémas de discrétisation et les algorithmes en termes de localité (spatiale et temporelle) et de vectorisation, vers des méthodes si possible plus vectorielles et plus denses.