Multilevel Monte Carlo

A few words on concurrency in C++11

Vincent Lemaire

LPMA – UPMC

June 11, 2015

Abstract linear simulation framework

- $Y_0 \in \mathbf{L}^2(\mathbf{P})$ not simulatable real-valued random variable.
- Aim: compute $I_0 = \mathbf{E}[Y_0]$ with a given accuracy $\varepsilon > 0$.

- Suppose we have family $(Y_h)_{h\in\mathcal{H}}$ of real-valued random variables in $\mathbf{L}^2(\mathbf{P})$ satisfying:
  - Bias error expansion (weak error expansion):

$$\exists\, \alpha > 0,\ \mathbf{E}[Y_h] = \mathbf{E}[Y_0] + c_1 h^{\alpha} + c_2 h^{2\alpha} + \cdots + c_{_R} h^{\alpha R}\big(1 + \eta_R(h)\big) \quad (WE_{\alpha,R})$$

  - Strong approximation error assumption:

$$\exists\, \beta > 0, \quad \big\|Y_h - Y_0\big\|_2^2 = \mathbf{E}\Big[\big|Y_h - Y_0\big|^2\Big] \leqslant V_1 h^{\beta} \qquad (SE_{\beta})$$

- h is the *bias parameter* taking value in $\mathcal{H} = \big\{\mathbf{h}/n, n \geqslant 1\big\}$.

Monte Carlo type estimators

We consider Monte Carlo type estimators I_π^N depending on some parameters π to compute

$$I_0 = \mathbf{E}\left[Y_0\right]$$

i.e. satisfying

- Constant bias: $\mathbf{E}\left[I_\pi^N\right] = \mathbf{E}\left[I_\pi^1\right]$ for all $N \geqslant 1$
- Inverse linear variance: $\mathrm{var}(I_\pi^N) = \frac{\nu(\pi)}{N}$ where $\nu(\pi) = \mathrm{var}(I_\pi^1)$
- Linear cost: $\mathrm{Cost}(I_\pi^N) = N\,\kappa(\pi)$ where $\kappa(\pi) = \mathrm{Cost}(I_\pi^1)$

Typically I_π^N is the N-empirical mean of i.d.d. r.v.

Aim: minimize the global cost of the estimator for a given error $\varepsilon > 0$.

$$(\pi(\varepsilon), N(\varepsilon)) = \underset{\left\|I_\pi^N - I_0\right\|_2 \leqslant \varepsilon}{\mathrm{argmin}} \mathrm{Cost}(I_\pi^N).$$

Definition

The *effort* of the estimator I_π^N is defined for every $\pi \in \Pi$ by

$$\phi(\pi) = \nu(\pi)\,\kappa(\pi).$$

- If the estimator I_π^N is *unbiased* then $\left\|I_\pi^1 - I_0\right\|_2^2 = \nu(\pi)$ so that

$$\pi(\varepsilon) = \pi_* = \operatorname*{argmin}_{\pi\in\Pi} \phi(\pi), \quad N(\varepsilon) = \frac{\nu(\pi_*)}{\varepsilon^2} = \frac{\phi(\pi_*)}{\kappa(\pi_*)\varepsilon^2}.$$

- If the estimator I_π^N is *biased* then $\left\|I_\pi^1 - I_0\right\|_2^2 = \nu(\pi) + \mu^2(\pi)$ where

$$\mu^2(\pi) = \left(\mathbf{E}\left[I_\pi^N\right] - I_0\right)^2 = \left(\mathbf{E}\left[I_\pi^1\right] - I_0\right)^2$$

so that

$$\pi(\varepsilon) = \operatorname*{argmin}_{\pi\in\Pi,\,|\mu(\pi)|<\varepsilon} \left(\frac{\phi(\pi)}{\varepsilon^2 - \mu^2(\pi)}\right), \; N(\varepsilon) = \frac{\phi(\pi(\varepsilon))}{\kappa(\pi(\varepsilon))(\varepsilon^2 - \mu^2(\pi(\varepsilon)))}.$$

Theorem (Crude Monte Carlo, $\pi = h$)

The Monte Carlo estimator of $\mathbf{E}[Y_0]$ *defined by* $\bar{Y}_h^N = \frac{1}{N}\sum_{k=1}^{N} Y_h^k$ *satisfies*

$$\mu(h) \sim c_1 h^\alpha, \quad \kappa(h) = \frac{1}{h}, \quad \phi(h) = \frac{\mathrm{var}(Y_h)}{h}.$$

Then, the optimal parameters $h^*(\varepsilon)$ *and* $N^*(\varepsilon)$ *are*

$$h^*(\varepsilon) = \frac{(1+2\alpha)^{-\frac{1}{2\alpha}}}{|c_1|^{1/\alpha}}\varepsilon^{\frac{1}{\alpha}}, \; N^*(\varepsilon) = \left(1+\frac{1}{2\alpha}\right)\frac{\mathrm{var}(Y_0)(1+\theta h^*(\varepsilon)^{\frac{\beta}{2}})^2}{\varepsilon^2}.$$

where $\theta = \sqrt{\frac{V_1}{\mathrm{var}(Y_0)}}$. *Furthermore, we have*

$$\limsup_{\varepsilon\to 0} \varepsilon^{2+\frac{1}{\alpha}} \min_{\substack{h\in\mathcal{H},\\ |\mu(h)|<\varepsilon}} \mathrm{Cost}(\bar{Y}_h^N) \leqslant |c_1|^{\frac{1}{\alpha}} \left(1+\frac{1}{2\alpha}\right)(1+2\alpha)^{\frac{1}{2\alpha}}\,\mathrm{var}(Y_0).$$

Multistep Richardson-Romberg estimator [P. 2007, MCMA]

- Aim: Kill the bias using the weak error expansion $(WE_{\alpha,R})$
- *Refiners:* R-tuple of integers $\underline{n} := (n_1, n_2, \ldots, n_R) \in \mathbf{N}^R$ satisfying

$$n_1 = 1 < n_2 < \cdots < n_R.$$

- We consider $Y_{h,\underline{n}}$ the $\mathbf{R}^R$-valued random vector

$$Y_{h,\underline{n}} = \left(Y_h, Y_{\frac{h}{n_2}}, \ldots, Y_{\frac{h}{n_R}} \right)$$

- We define the vector $\mathbf{w} = (\mathbf{w}_1, \ldots, \mathbf{w}_R)$ as the unique solution to the Vandermonde system $V\,\mathbf{w} = e_1$ where

$$V = V(1, n_2^{-\alpha}, \ldots, n_R^{-\alpha}) = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & n_2^{-\alpha} & \cdots & n_R^{-\alpha} \\ \vdots & \vdots & \cdots & \vdots \\ 1 & n_2^{-\alpha(R-1)} & \cdots & n_R^{-\alpha(R-1)} \end{pmatrix}.$$

- Key point: The solution $\mathbf{w}$ has a closed form given by Cramer's rule:

$$\forall i \in \{1, \dots, R\}, \quad \mathbf{w}_i = \frac{(-1)^{R-i} n_i^{\alpha(R-1)}}{\prod_{1 \leqslant j < i} (n_i^\alpha - n_j^\alpha) \prod_{i < j \leqslant R} (n_j^\alpha - n_i^\alpha)}.$$

- We also derive the following identity of interest in what follows

$$\widetilde{\mathbf{w}}_{R+1} := \sum_{i=1}^{R} \frac{\mathbf{w}_i}{n_i^{\alpha R}} = \frac{(-1)^{R-1}}{\underline{n}!^\alpha}.$$

Multistep Richardson-Romberg estimator

Definition (Multistep Richardson-Romberg estimator)

$$\bar{Y}^N_{h,\underline{n}} = \frac{1}{N}\sum_{k=1}^{N} \left\langle \mathbf{w}, Y^k_{h,\underline{n}} \right\rangle, \quad \left(Y^k_{h,\underline{n}}\right)_{k\geqslant 1} \text{ i.i.d.}$$

$$= \frac{1}{N}\sum_{k=1}^{N}\sum_{i=1}^{R} \mathbf{w}_i \, Y^k_{\frac{h}{n_i}}$$

Note that $\pi = (h, \underline{n}, R)$ and

$$\mu(\pi) = \mathbf{E}\left[\left\langle \mathbf{w}, Y^1_{h,\underline{n}} \right\rangle\right] = \left\langle \mathbf{w}, \mathbf{E}\left[Y^1_{h,\underline{n}}\right]\right\rangle$$

$$= \mathbf{E}\left[Y_0\right] + \widetilde{\mathbf{w}}_{R+1} c_R h^{\alpha R}(1 + \eta_{R,\underline{n}}(h))$$

Let $|\underline{n}| = n_1 + \cdots + n_R$ and $\underline{n}! = n_1 \cdots n_R$.

Theorem (Multistep Richardson-Romberg)

The Multistep Richardson-Romberg estimator of $\mathbf{E}[Y_0]$ *satisfies*

$$\mu(h) \simeq (-1)^{R-1} c_R \left(\frac{h^R}{\underline{n}!}\right)^{\alpha}, \ \kappa(h) = \frac{|\underline{n}|}{h}, \ \phi(h) \simeq \frac{|\underline{n}| \operatorname{var}(Y_0)}{h}$$

and for a fixed $R \geqslant 2$ *the optimal parameters* $h^*(\varepsilon)$ *and* $N^*(\varepsilon)$ *are*

$$h^*(\varepsilon) = \frac{(1+2\alpha R)^{-\frac{1}{2\alpha R}}}{|c_R|^{1/(\alpha R)}} \varepsilon^{\frac{1}{\alpha R}}, \ N^*(\varepsilon) = \left(1 + \frac{1}{2\alpha R}\right) \frac{\operatorname{var}(Y_0)(1 + \theta\, h^*(\varepsilon)^{\frac{\beta}{2}})^2}{\varepsilon^2}.$$

Furthermore, we have

$$\inf_{\substack{h \in \mathcal{H} \\ |\mu(h)| < \varepsilon}} \operatorname{Cost}(\bar{Y}_h^N) \sim \left(\frac{(1+2\alpha R)^{1+\frac{1}{2\alpha R}}}{2\alpha R}\right) \frac{|c_R|^{\frac{1}{\alpha R}} \operatorname{var}(Y_0)}{\varepsilon^{2+\frac{1}{\alpha R}}} \frac{|\underline{n}|}{\underline{n}!^{\frac{1}{R}}} \quad \text{as} \ \ \varepsilon \to 0.$$

Remarks on the Multistep estimator

- If $n_i = i$ then $\left|\underline{n}\right| = \frac{R(R+1)}{2}$ and $(n!)^{\frac{1}{R}} \sim \frac{R}{e}$ so that $\frac{\left|\underline{n}\right|}{n!^{\frac{1}{R}}} \sim \frac{e(R+1)}{2}$.
- If $n_i = M^{i-1}$ $(M \geqslant 2)$ then $\frac{\left|\underline{n}\right|}{\underline{n}!^{\frac{1}{R}}} \sim M^{\frac{R-1}{2}}$

How to avoid this drawback ?

Using control variance technique such as Multilevel:

- We consider R *independant* copies $(Y_{h,\underline{n}}^{(j)})$, $j = 1, \ldots, R$ of the random vector $Y_{h,\underline{n}}$ and we *split* $\left\langle \mathbf{w}, Y_{h,\underline{n}} \right\rangle$ into

$$\left\langle \mathbf{w}, Y_{h,\underline{n}} \right\rangle \rightsquigarrow \sum_{j=1}^{R} \left\langle \mathbf{T}^j, Y_{h,\underline{n}}^{(j)} \right\rangle = \sum_{i,j=1}^{R} \mathbf{T}_i^j \, Y_{\frac{h}{n_i}}^{(j)}$$

where $\mathbf{T} = [\mathbf{T}^1 \ldots \mathbf{T}^R]$ is an $R \times R$ matrix such that $\sum_j \mathbf{T}^j = \mathbf{w}$.

Definitions

- Allocation matrix: An $R \times R$-matrix $\mathbf{T} = [\mathbf{T}^1 \dots \mathbf{T}^R]$ satisfying

$$\mathbf{T}^1 = e_1 \quad \text{and} \quad \forall\, j \in \{2, \dots, R\},\ \sum_{i=1}^{R} \mathbf{T}_i^j = 0. \tag{1}$$

  so that $$\sum_{i,j=1}^{d} \mathbf{T}_i^j = 1.$$

- Stratification/Allocation strategy: Let $q = (q_1, \dots, q_R)$, $q_j > 0$, $j = 1, \dots, R$ and $\sum_j q_j = 1$. Let $N_j = \lceil q_j N \rceil$.

- Multilevel estimator of order R

$$\bar{Y}_{h,\underline{n}}^{N,q} = \sum_{j=1}^{R} \frac{1}{N_j} \sum_{k=1}^{N_j} \left\langle \mathbf{T}^j, Y_{h,\underline{n}}^{(j),k} \right\rangle \approx \frac{1}{N} \sum_{j=1}^{R} \frac{1}{q_j} \sum_{k=1}^{N_j} \left\langle \mathbf{T}^j, Y_{h,\underline{n}}^{(j),k} \right\rangle \tag{2}$$

  - Multilevel Monte Carlo $\equiv \sum_{j=1}^{R} \mathbf{T}^j = e_R$
  - Multilevel Richardson-Romberg $\equiv \sum_{j=1}^{R} \mathbf{T}^j = \mathbf{w}$

Effort of a Multilevel estimator

- Parameters: $\pi = (\pi_0, q)$ with $\pi_0 = (h, \underline{n}, R, \mathbf{T})$.
- (Unitary) Cost of a Multilevel estimator

$$\mathrm{Cost}(\bar{Y}^{N,q}_{h,\underline{n}}) = \sum_{j=1}^{R} N_j \sum_{i=1}^{R} \frac{1}{h} n_i \mathbf{1}_{\{\mathbf{T}^j_i \neq 0\}} = N\, \kappa(\pi)$$

where the unitary complexity $\kappa(\pi)$ is given by

$$\kappa(\pi) = \frac{1}{h} \sum_{j=1}^{R} q_j \sum_{i=1}^{R} n_i \mathbf{1}_{\{\mathbf{T}^j_i \neq 0\}}.$$

- Effort of a Multilevel estimator:

$$\phi(\pi) = \nu(\pi)\, \kappa(\pi) = \left(\sum_{j=1}^{R} \frac{1}{q_j} \mathrm{var}\left(\left\langle \mathbf{T}^j, Y^{(j)}_{h,\underline{n}} \right\rangle \right) \right) \kappa(\pi)$$

Theorem (Bias)

Assume $(WE_{\alpha,R})$ *holds.*

(i) *Multilevel Richardson-Romberg estimator*

$$\mu(\pi_0, q) \simeq (-1)^{R-1} c_R \left(\frac{h^R}{\underline{n}!} \right)^{\alpha}$$

(ii) *Multilevel Monte Carlo estimator*

$$\mu(\pi_0, q) \simeq c_1 \left(\frac{h}{n_R} \right)^{\alpha}$$

In both cases, the bias does not depend on the stratification strategy q selected in the simplex $\mathcal{S}_+(R)$.

Steps to minimize the global cost

Step 1: Minimize the effort ϕ over admissible stratification strategies

$$q^* = q^*(\pi_0) = \underset{q \in \mathcal{S}_+(R)}{\operatorname{argmin}} \bar{\phi}(\pi_0, q) \quad \text{where} \quad \bar{\phi}(\pi) \geqslant \phi(\pi),$$

Let $\phi^*(\pi_0) = \phi(\pi_0, q^*)$ be the optimally stratified effort.

Step 2: Minimizing the resulting cost as a function of the remaining parameters π_0 for a given target error ε

$$\pi_0(\varepsilon) = \underset{\substack{\pi_0 \in \Pi_0 \\ |\mu(\pi_0, q^*)| < \varepsilon}}{\operatorname{argmin}} \left(\frac{\phi^*(\pi_0)}{\varepsilon^2 - \mu^2(\pi_0, q^*)} \right),$$

$$N(\pi_0(\varepsilon)) = \frac{\phi^*(\pi_0(\varepsilon))}{\kappa(\pi_0(\varepsilon), q^*)(\varepsilon^2 - \mu^2(\pi_0, q^*))}.$$

Optimal stratification (no weak error expansion needed)

Theorem

Assume (SE_β) *holds, and let* $\theta = \sqrt{\frac{V_1}{\mathrm{var}(Y_0)}}$. *Then the optimally stratified effort* ϕ^* *satisfies*

$$\phi^*(\pi_0) \leqslant \frac{\mathrm{var}(Y_0)}{h}\left(1+\theta h^{\frac{\beta}{2}}\sum_{j=1}^{R}\left(\sum_{i=1}^{R}\left|\mathbf{T}_i^j\right| n_i^{-\frac{\beta}{2}}\right)\left(\sum_{i=1}^{R} n_i \mathbf{1}_{\left\{\mathbf{T}_i^j \neq 0\right\}}\right)^{\frac{1}{2}}\right)^2$$

where the optimal stratification strategy $q^* = q^*(\pi_0)$ *is given by*

$$\begin{cases} q_1^*(\pi_0) = \mu^*(1+\theta h^{\frac{\beta}{2}}) \\ q_j^*(\pi_0) = \mu^* \theta h^{\frac{\beta}{2}}\left(\sum_{i=1}^{R}\left|\mathbf{T}_i^j\right| n_i^{-\frac{\beta}{2}}\right)\left(\sum_{i=1}^{R} n_i \mathbf{1}_{\left\{\mathbf{T}_i^j \neq 0\right\}}\right)^{-\frac{1}{2}}, \ j = 2, \ldots, R, \end{cases} \tag{3}$$

with μ^* *is the normalizing constant such that* $\sum_{j=1}^{R} q_j^* = 1$.

ML2R

Among several possibilities we specify the allocation matrix $\mathbf{T}$ as follows:

$$\mathbf{T} = \begin{pmatrix} \mathbf{W}_1 & -\mathbf{W}_2 & 0 & \cdots & \cdots & 0 \\ 0 & \mathbf{W}_2 & -\mathbf{W}_3 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & \cdots & 0 & \mathbf{W}_{R-1} & -\mathbf{W}_R \\ 0 & \cdots & \cdots & \cdots & 0 & \mathbf{W}_R \end{pmatrix}.$$

where

$$\mathbf{W}_j = \sum_{k=j}^{R} \mathbf{w}_k, \ j = 1, \ldots, R$$

so that $\mathbf{W}_1 = 1$.

MLMC

$$\mathbf{T} = \begin{pmatrix} 1 & -1 & 0 & \cdots & \cdots & 0 \\ 0 & 1 & -1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & \cdots & 0 & 1 & -1 \\ 0 & \cdots & \cdots & \cdots & 0 & 1 \end{pmatrix}.$$

Let $R \geqslant 2$ and the n_i, $i = 1, \dots, R$ be fixed parameters.

Theorem (Bias parameter optimization (ML2R))

Assume $(WE_{\alpha,R})$ *and* (SE_β) *hold.*
A Multilevel Richardson-Romberg estimator satisfies

$$\inf_{\substack{h \in \mathcal{H} \\ |\mu(h,q^*)|<\varepsilon}} \mathrm{Cost}\left(\bar{Y}_{h,\underline{n}}^{N,q^*}\right) \sim \left(\frac{(1+2\alpha R)^{1+\frac{1}{2\alpha R}}}{2\alpha R}\right) \frac{|c_R^{\frac{1}{\alpha R}}|\,\mathrm{var}(Y_0)}{\varepsilon^{2+\frac{1}{\alpha R}}} \frac{1}{\underline{n}!^{\frac{1}{R}}} \quad \text{as } \varepsilon \to 0,$$

with q^* *defined by* (3). *Note that* $|\underline{n}|$ *disappeared...*
This asymptotically optimal bound is achieved with the bias parameter:

$$h^*(\varepsilon, R) = (1+2\alpha R)^{-\frac{1}{2\alpha R}} \left(\frac{\varepsilon}{|c_R|}\right)^{\frac{1}{\alpha R}} \underline{n}!^{\frac{1}{R}}. \tag{4}$$

Let $R \geqslant 2$ and let n_i, $i = 1, \ldots, R$ be fixed parameters.

Theorem (Bias parameter optimization (MLMC))

Assume $(WE_{\alpha,R})$ *and* (SE_β) *hold.*
A Multilevel Monte Carlo estimator satisfies

$$\inf_{\substack{h \in \mathcal{H} \\ |\mu(h,q^*)|<\varepsilon}} \mathrm{Cost}\left(\bar{Y}_{h,\underline{n}}^{N,q^*}\right) \sim \left(\frac{(1+2\alpha)^{1+\frac{1}{2\alpha}}}{2\alpha}\right) \frac{|c_1|^{\frac{1}{\alpha}}\,\mathrm{var}(Y_0)}{n_{_R}\varepsilon^{2+\frac{1}{\alpha}}} \qquad \text{as } \varepsilon \to 0,$$

with q^* *defined by* (3).
This asymptotically optimal bound is achieved with a bias parameter given by

$$h^*(\varepsilon, R) = (1+2\alpha)^{-\frac{1}{2\alpha}} \left(\frac{\varepsilon}{|c_1|}\right)^{\frac{1}{\alpha}} n_{_R}. \tag{5}$$

Theorem (Main (ML2R) result when $n_i = M^{i-1}$, $i = 1, \dots, R$.)

Assume $\lim_R |c_R|^{1/R} = \widetilde{c} \in (0, +\infty)$. *The Multilevel Richardson-Romberg estimator satisfies*

$$\overline{\lim_{\varepsilon \to 0}} \, v(\beta, \varepsilon) \times \inf_{\substack{h \in \mathcal{H}, R \geqslant 2 \\ |\mu(h,R,q^*)| < \varepsilon}} \mathrm{Cost}\left(Y_{h,\underline{n}}^{N,q}\right) \leqslant K(\alpha, \beta, M)$$

with

$$v(\beta, \varepsilon) = \begin{cases} \varepsilon^2 \left(\log(1/\varepsilon)\right)^{-1} & \text{if } \beta = 1, \\ \varepsilon^2 & \text{if } \beta > 1, \\ \varepsilon^2 e^{-\frac{1-\beta}{\sqrt{\alpha}} \sqrt{2 \log(1/\varepsilon) \log(M)}} & \text{if } \beta < 1. \end{cases}$$

These bounds are achieved with an order

$$R^*(\varepsilon) = \left\lfloor \frac{1}{2} + \frac{\log(\mathbf{h}\, \widetilde{c}^{\frac{1}{\alpha}})}{\log(M)} + \sqrt{\left(\frac{1}{2} + \frac{\log(\mathbf{h}\, \widetilde{c}^{\frac{1}{\alpha}})}{\log(M)}\right)^2 + 2 \frac{\log(A/\varepsilon)}{\alpha \log(M)}} \right\rfloor$$

$A = \sqrt{1 + 4\alpha}$, *and the bias parameter* $h^*(\varepsilon, R(\varepsilon)) \in (0, \mathbf{h}]$ *is given by* (4).

Theorem (Main result (MLMC))

The Multilevel Monte Carlo estimator satisfies

$$\overline{\lim_{\varepsilon\to 0}}\, v(\beta,\varepsilon) \times \inf_{\substack{h\in\mathcal{H}, R\geqslant 2 \\ |\mu(h,R,q^*)|<\varepsilon}} \mathrm{Cost}\left(Y_{h,\underline{n}}^{N,q}\right) \leqslant K(\alpha,\beta,M)$$

$$\text{with} \quad v(\beta,\varepsilon) = \begin{cases} \varepsilon^2 (\log(1/\varepsilon))^{-2} & \text{if } \beta = 1 \\ \varepsilon^2 & \text{if } \beta > 1 \\ \varepsilon^{2+\frac{1-\beta}{\alpha}} & \text{if } \beta < 1. \end{cases}$$

These bounds are achieved with an order

$$R^*(\varepsilon) = \left\lfloor 1 + \frac{\log\left(|c_1|^{\frac{1}{\alpha}}\mathbf{h}\right)}{\log(M)} + \frac{\log(A/\varepsilon)}{\alpha\log(M)} \right\rfloor,$$

$A = \sqrt{1+2\alpha}$, *and a bias parameter* $h^*(\varepsilon, R(\varepsilon)) \in (0, \mathbf{h}]$ *given by* (5).

Optimal parameters for ML2R (with $n_i = M^{i-1}$)

$$R \quad \left| \quad \left\lfloor \frac{1}{2} + \frac{\log(\widetilde{c}^{\frac{1}{\alpha}}\mathbf{h})}{\log(M)} + \sqrt{\left(\frac{1}{2} + \frac{\log(\widetilde{c}^{\frac{1}{\alpha}}\mathbf{h})}{\log(M)}\right)^2 + 2\frac{\log(A/\varepsilon)}{\alpha\log(M)}} \right\rfloor, \; A = \sqrt{1+4\alpha}$$

$$h^{-1} \quad \left| \quad \left\lceil (1+2\alpha R)^{\frac{1}{2\alpha R}} \varepsilon^{-\frac{1}{\alpha R}} M^{-\frac{R-1}{2}} \right\rceil \right.$$

$$q \quad \left| \quad \begin{aligned} q_1 &= \mu^*(1+\theta h^{\frac{\beta}{2}}) \\ q_j &= \mu^* \theta h^{\frac{\beta}{2}} \left(\left|\mathbf{W}_j(R,M)\right| \frac{n_{j-1}^{-\frac{\beta}{2}} + n_j^{-\frac{\beta}{2}}}{\sqrt{n_{j-1}+n_j}} \right), \end{aligned} \right.$$

$$N \quad \left| \quad \frac{\mathrm{var}(Y_0)\left(1 + \theta h^{\frac{\beta}{2}} \sum_{j=1}^{R} \left|\mathbf{W}_j(R,M)\right| \left(n_{j-1}^{-\frac{\beta}{2}} + n_j^{-\frac{\beta}{2}}\right) \sqrt{n_{j-1}+n_j}\right)^2}{\varepsilon^2 \left(1+\frac{1}{2\alpha R}\right)^{-1} \sum_{j=1}^{R} q_j (n_{j-1}+n_j)} \right.$$

Optimal parameters for MLMC (with $n_i = M^{i-1}$)

R	$\left\lfloor 1 + \dfrac{\log(\lvert c_1 \rvert^{\frac{1}{\alpha}} \mathbf{h})}{\log(M)} + \dfrac{\log(A/\varepsilon)}{\alpha \log(M)} \right\rfloor, \; A = \sqrt{1+2\alpha}$
h^{-1}	$\left\lceil (1+2\alpha)^{\frac{1}{2\alpha}} \varepsilon^{-\frac{1}{\alpha}} \lvert c_1 \rvert^{\frac{1}{\alpha}} M^{-(R-1)} \right\rceil$
q	$q_1 = \mu^*(1+\theta h^{\frac{\beta}{2}})$ $q_j = \mu^* \theta h^{\frac{\beta}{2}} \left(\dfrac{n_{j-1}^{-\frac{\beta}{2}} + n_j^{-\frac{\beta}{2}}}{\sqrt{n_{j-1}+n_j}} \right), \quad$
N	$\dfrac{\mathrm{var}(Y_0) \left(1 + \theta h^{\frac{\beta}{2}} \sum_{j=1}^{R} \left(n_{j-1}^{-\frac{\beta}{2}} + n_j^{-\frac{\beta}{2}} \right) \sqrt{n_{j-1}+n_j} \right)^2}{\varepsilon^2 \left(1 + \dfrac{1}{2\alpha} \right)^{-1} \sum_{j=1}^{R} q_j (n_{j-1}+n_j)}$

Choice of the root $M = M(\varepsilon)$

- Refiners: $n_i = M^{i-1}$, $i = 1, \ldots, R$, M integer, $M \geqslant 2$.
- Optimal parameters are all functions of $M \Longrightarrow$ Numerical optimization of the cost fonction

$$\mathbf{C}_\varepsilon(M) = \mathrm{Cost}(Y_{h(M),\underline{n}}^{N(M),q(M)}) := N(M)\,\kappa(h(M), R(M), q(M)),$$

for $M \in \{2, \ldots, 10\}$, where

$$\kappa(h, R, q) = \frac{1}{h} \sum_{j=1}^{R} q_j (n_{j-1} + n_j)$$

(with $n_0 = 0$) or (for nested simulations)

$$\kappa(h, R, q) = \frac{1}{h} \sum_{j=1}^{R} q_j n_j.$$

Nested Monte Carlo

- Aim: Computing
$$\mathbf{E}\left[\varphi\big(\mathbf{E}\left[X \mid Y\right]\big)\right]$$
where (X, Y) is couple of $\mathbf{R} \times \mathbf{R}^{q_Y}$-valued random variable.
- Assume that X has the representation
$$X = F(Z, Y).$$
where: $F : \mathbf{R}^{q_Z} \times \mathbf{R}^{q_Y} \to \mathbf{R}$ is a Borel function
$Z : (\Omega, \mathcal{A}) \to \mathbf{R}^{q_Z}$ is independent of Y.

To comply with the multilevel framework, we set $\mathcal{H} = \{1/K,\, K \geqslant 1\}$

$$Y_0 = \varphi\big(\mathbf{E}\left[X \mid Y\right]\big), \quad Y_{\frac{1}{K}} = \varphi\left(\frac{1}{K}\sum_{k=1}^{K} F(Z^k, Y)\right).$$

Examples of Financial applications

- Compound option pricing
- CVA (credit value adjustment) computation
- Solvency capital requirement (SCR) computation

Some references:

- Gordy & Juneja, *Nested simulation in portfolio risk measurement*, 2010
- Broadie et al. *Efficient risk estimation via nested sequential simulation*, 2011
- Ghamami & Zhang *Efficient Monte Carlo Counterparty Credit Risk Pricing and Measurement*, 2013

Nested Monte Carlo

Lemma (Strong approximation)

Assume φ is Lipschitz continuous. For every $h \in \mathcal{H}$,

$$\left\|Y_0 - Y_h\right\|_2^2 \leqslant [f]_{\mathrm{Lip}}^2 \left(\left\|X\right\|_2^2 - \left\|\mathbf{E}\left[X \mid Y\right]\right\|_2^2\right) h \tag{6}$$

so that $(Y_h)_{h\in\mathcal{H}}$ satisfies (SE_β) with $\beta = 1$.

Lemma (Bias error (I))

Let $R \geqslant 1$ and let $\varphi : \mathbf{R} \to \mathbf{R}$ be a $2R+1$ times differentiable function with $\varphi^{(k)}$, $k = R+1, \ldots, 2R+1$, bounded over the real line. Assume $X \in L^{2R+1}$. Then there exists $c_1, \ldots, c_R$ such that

$$\forall h \in \mathcal{H}, \quad \mathbf{E}\left[\varphi(Y_h)\right] = \mathbf{E}\left[\varphi(Y_0)\right] + \sum_{r=1}^{R} c_r h^r + \mathcal{O}(h^{R+1/2}). \tag{7}$$

Nested Monte Carlo

Let f_{Y_0} be the density of Y_0 and let $f_{Y_0|Y_h-Y_0=\xi}$ be the density of Y_0 given the occurence of the value ξ of $Y_h - Y_0$.

Lemma (Bias error (II), work in progress)

Let $R \geqslant 1$. Assume that the functions f_{Y_0}, $f_{Y_0|Y_h-Y_0=\xi}$, for $\xi \in \mathbf{R}$, are $2R+1$ times differentiable. Assume $X \in L^{2R+1}$ and a little more...

$$f_{Y_h}(y) = f_{Y_0}(y) + f_{Y_0}(y) \sum_{r=1}^{R} \frac{h^r}{r!} P_r(y) + o(h^{R+\frac{1}{2}}) \tag{8}$$

uniformly in $y \in \mathbf{R}^{q_Y}$, where P_r is an explicit function with polynomial growth.

Put-on-call option

- Payoff of a Put-on-Call option (involves conditional expectation)

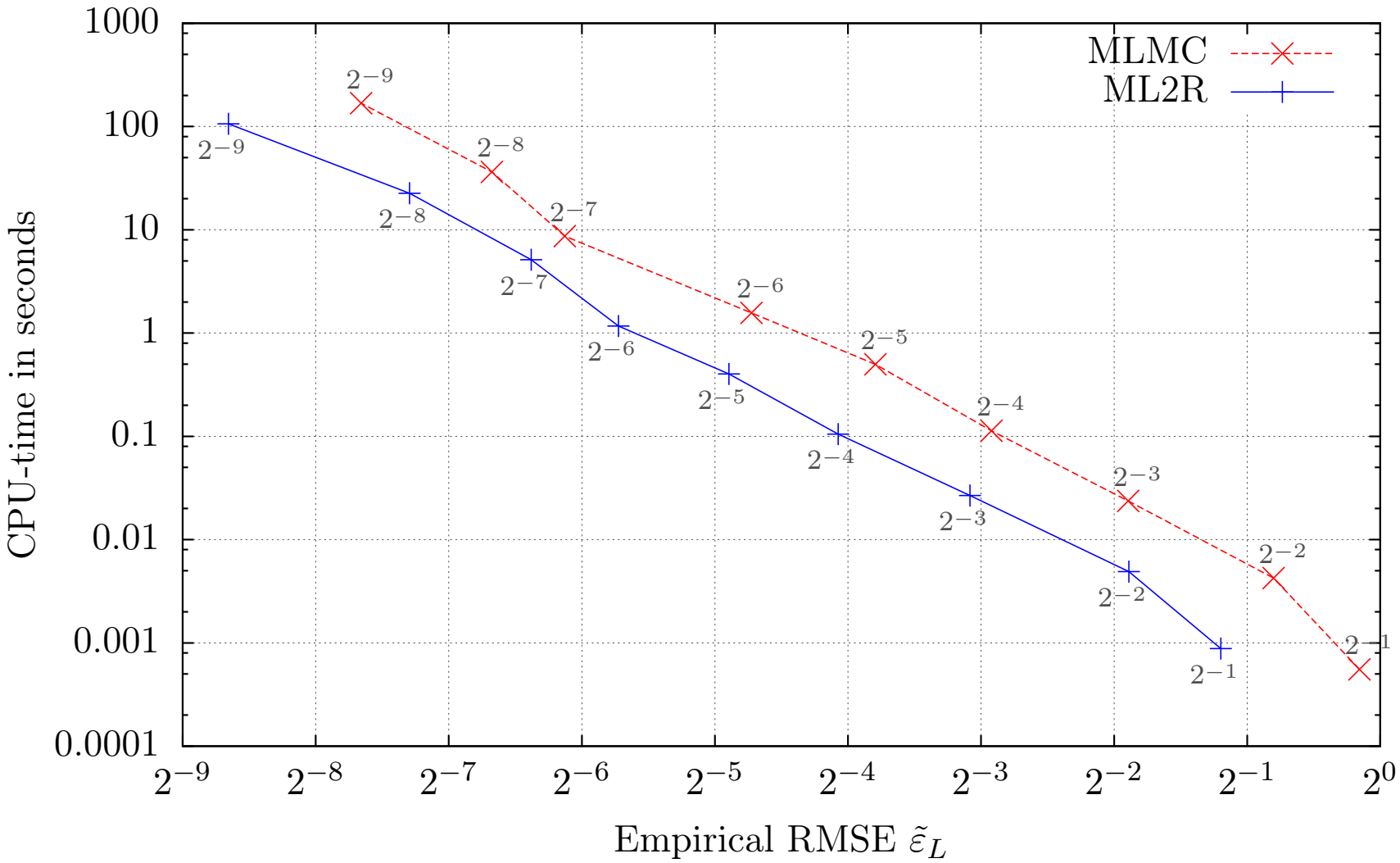
$$(K_1 - \mathbf{E}\left[(S_{T_2} - K_2)_+ \mid S_{T_1}\right])_+$$

with parameters $T_1 = 1/12$, $T_2 = 1/2$ and $K_1 = 6.5$, $K_2 = 100$.

- Black-Scholes model with $s_0 = 100$, $r = 0.03$ and $\sigma = 0.3$.
Note that

$$S_{T_2} = F(G, S_{T_1}) = S_{T_1} e^{(r - \frac{\sigma^2}{2})(T_2 - T_1) + \sigma\sqrt{T_2 - T_1}G}$$

CPU-time as a function of $\tilde{\varepsilon}_L$



Parameters and complexity for a given ε

		$ML2R$				$MLMC$			
k	$\varepsilon = 2^{-k}$	R	M	h^{-1}	Cost	R	M	h^{-1}	Cost
1	$5.00 \cdot 10^{-01}$	2	5	1	$1.37 \cdot 10^{+03}$	2	4	1	$1.14 \cdot 10^{+03}$
2	$2.50 \cdot 10^{-01}$	2	9	1	$6.33 \cdot 10^{+03}$	2	7	1	$5.76 \cdot 10^{+03}$
3	$1.25 \cdot 10^{-01}$	3	3	1	$4.65 \cdot 10^{+04}$	3	4	1	$4.57 \cdot 10^{+04}$
4	$6.25 \cdot 10^{-02}$	3	4	1	$1.87 \cdot 10^{+05}$	3	6	1	$2.26 \cdot 10^{+05}$
5	$3.12 \cdot 10^{-02}$	3	5	1	$7.84 \cdot 10^{+05}$	3	8	1	$1.06 \cdot 10^{+06}$
6	$1.56 \cdot 10^{-02}$	3	6	1	$3.32 \cdot 10^{+06}$	4	5	1	$6.21 \cdot 10^{+06}$
7	$7.81 \cdot 10^{-03}$	3	7	1	$1.41 \cdot 10^{+07}$	4	7	1	$3.02 \cdot 10^{+07}$
8	$3.91 \cdot 10^{-03}$	3	9	1	$6.28 \cdot 10^{+07}$	4	8	1	$1.31 \cdot 10^{+08}$
9	$1.95 \cdot 10^{-03}$	4	4	1	$3.26 \cdot 10^{+08}$	4	10	1	$6.06 \cdot 10^{+08}$

Server system Pharaon2

- Nom: `pharaon2.lpma.math.upmc.fr`
- Adresse IP: `134.157.65.205`
- `Linux 2.6.32_64 #1 SMP`

```
CPU(s):                64
Thread(s) per core:    2
Core(s) per socket:    8
Socket(s):             4
NUMA node(s):          4
Model name:            Intel(R) Xeon(R) CPU E5-4620 0 @ 2.20GHz
CPU MHz:               1200.000
BogoMIPS:              4399.44
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              16384K
```

Server system Pharaon2

- Nom: `pharaon2.lpma.math.upmc.fr`
- Adresse IP: `134.157.65.205`
- `Linux 2.6.32_64 #1 SMP`

```
CPU(s):                64
Thread(s) per core:    2
Core(s) per socket:    8
Socket(s):             4
NUMA node(s):          4
Model name:            Intel(R) Xeon(R) CPU E5-4620 0 @ 2.20GHz
CPU MHz:               1200.000
BogoMIPS:              4399.44
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              16384K
```

Desktop system Nephtys

```
CPU(s):                8
Thread(s) per core:    2
Core(s) per socket:    4
Socket(s):             1
NUMA node(s):          1
Model name:            Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz
CPU MHz:               1600.125
CPU max MHz:           3900.0000
CPU min MHz:           1600.0000
BogoMIPS:              6803.85
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              8192K
```

Threads

A *thread of execution* is a sequence of instructions that can be executed concurrently with other such sequences in multithreading environments, while sharing a same address space.

```
template <class Fn, class... Args>
explicit thread(Fn && fn, Args &&... args);
```

- Construct a thread object that represents a new joinable thread of execution.
- The new thread of execution calls `fn` passing `args` as arguments.
- The arguments to the thread function are moved or copied by value. If a reference argument needs to be passed to the thread function, it has to be wrapped (e.g. with `std::ref` or `std::cref`).
- Any return value from the function `fn` is ignored.

Class `std::thread`

- Public member class

`id`	represents the id of a thread

- Public member functions

Constructor	
Destructor	
`operator=`	moves the thread object
`joinable`	checks whether the thread is joinable, i.e. potentially running in parallel context
`get_id`	returns the id of the thread
`hardware_concurrency`	returns the number of concurrent threads supported by the implementation
`join`	waits for a thread to finish its execution
`detach`	permits the thread to execute independently from the thread handle
`swap`	swaps two thread objects

Class `std::thread`

Static to get the number of hardware threads

- Public member class

`id`	represents the id of a thread

- Public member functions

Constructor	
Destructor	
`operator=`	moves the thread object
`joinable`	checks whether the thread is joinable, i.e. potentially running in parallel context
`get_id`	returns the id of the thread
`hardware_concurrency`	returns the number of concurrent threads supported by the implementation
`join`	waits for a thread to finish its execution
`detach`	permits the thread to execute independently from the thread handle
`swap`	swaps two thread objects

Class `std::thread`

Main function of the class `std::thread`

- Public member class

`id`	represents the id of a thread

- Public member functions

Constructor	
Destructor	
`operator=`	moves the thread object
`joinable`	checks whether the thread is joinable, i.e. potentially running in parallel context
`get_id`	returns the id of the thread
`hardware_concurrency`	returns the number of concurrent threads supported by the implementation
`join`	waits for a thread to finish its execution
`detach`	permits the thread to execute independently from the thread handle
`swap`	swaps two thread objects

Initialization of the generators

```
using clock_type = std::chrono::high_resolution_clock;
using generator_type = std::mt19937_64;
unsigned hw_threads = std::thread::hardware_concurrency();

unsigned seed = clock_type::now().time_since_epoch().count();
std::seed_seq seq({seed, seed+1, seed+2, seed+3, seed+4});
std::vector<unsigned> seeds(hw_threads);
seq.generate(seeds.begin(), seeds.end());

std::vector<generator_type> gens;
for (unsigned i = 0; i < hw_threads; ++i)
    gens.push_back(generator_type(seeds[i]));
```

Initialization of the generators

Quasi random number generator: Mersenne Twister 64 bits

```
using clock_type = std::chrono::high_resolution_clock;
using generator_type = std::mt19937_64;
unsigned hw_threads = std::thread::hardware_concurrency();

unsigned seed = clock_type::now().time_since_epoch().count();
std::seed_seq seq({seed, seed+1, seed+2, seed+3, seed+4});
std::vector<unsigned> seeds(hw_threads);
seq.generate(seeds.begin(), seeds.end());

std::vector<generator_type> gens;
for (unsigned i = 0; i < hw_threads; ++i)
    gens.push_back(generator_type(seeds[i]));
```

Initialization of the generators

Initialization of vector seeds by random realizations

```
using clock_type = std::chrono::high_resolution_clock;
using generator_type = std::mt19937_64;
unsigned hw_threads = std::thread::hardware_concurrency();

unsigned seed = clock_type::now().time_since_epoch().count();
std::seed_seq seq({seed, seed+1, seed+2, seed+3, seed+4});
std::vector<unsigned> seeds(hw_threads);
seq.generate(seeds.begin(), seeds.end());

std::vector<generator_type> gens;
for (unsigned i = 0; i < hw_threads; ++i)
    gens.push_back(generator_type(seeds[i]));
```

Definition of the template class

```
template <typename TLinearEstimator>
class parallelize_estimator : public TLinearEstimator {
  public:
    parallelize_estimator(TLinearEstimator const & X,
                          unsigned nb_threads)
        : TLinearEstimator(X), _nb_threads(nb_threads) {
        for (unsigned i = 0; i < _nb_threads-1; ++i)
            _Xs.push_back(X);
    }
    parallelize_estimator & reinit() {
        TLinearEstimator::reinit();
        for (TLinearEstimator & est : _Xs) est.reinit();
        return *this;
    }
    parallelize_estimator & operator()(generator_type & first,
                                       unsigned M);
  private:
    std::vector<TLinearEstimator> _Xs;
    unsigned _nb_threads;
};
```

Definition of the template class

Definition of this method in the next slides

```
template <typename TLinearEstimator>
class parallelize_estimator : public TLinearEstimator {
  public:
    parallelize_estimator(TLinearEstimator const & X,
                          unsigned nb_threads)
        : TLinearEstimator(X), _nb_threads(nb_threads) {
        for (unsigned i = 0; i < _nb_threads-1; ++i)
            _Xs.push_back(X);
    }
    parallelize_estimator & reinit() {
        TLinearEstimator::reinit();
        for (TLinearEstimator & est : _Xs) est.reinit();
        return *this;
    }
    parallelize_estimator & operator()(generator_type & first,
                                       unsigned M);
  private:
    std::vector<TLinearEstimator> _Xs;
    unsigned _nb_threads;
};
```

Definition of the method operator()

```
template <typename TLinearEstimator>
parallelize_estimator<TLinearEstimator> &
parallelize_estimator<TLinearEstimator>::operator()(
        generator_type & first,
        unsigned M)
{
  std::list<std::thread> threads;
  unsigned q = M / _nb_threads;
  unsigned r = M % _nb_threads;
  unsigned i = 0;

  // suite sur le prochain slide...
```

Definition of the method operator()

```
template <typename TLinearEstimator>
parallelize_estimator<TLinearEstimator> &
parallelize_estimator<TLinearEstimator>::operator()(
        generator_type & first,
        unsigned M)
{
  std::list<std::thread> threads;
  unsigned q = M / _nb_threads;
  unsigned r = M % _nb_threads;
  unsigned i = 0;

  // suite sur le prochain slide...
```

To overload the **operator()** of the class TLinearEstimator. Assume that there exist nb_threads generator_type at adress &first

Definition of the method operator()

```
  while (i < _nb_threads-1) {
    _Xs[i].reinit();
    unsigned Mi = q + (i < r ? 1 : 0);
    threads.push_back(std::thread(std::ref(_Xs[i]),
                                  std::ref(*(&first+i)),
                                  Mi));
    ++i;
  }

  TLinearEstimator::operator()(*(&first+i),
                               q + (i < r ? 1 : 0));

  for (auto & th : threads) th.join();

  for (auto const & Y : _Xs) *this += Y;
  return *this;
};
```

Definition of the method operator

Creates new std::thread object and associates it with a thread of execution

```
  while (i < _nb_threads-1) {
    _Xs[i].reinit();
    unsigned Mi = q + (i < r ? 1 : 0);
    threads.push_back(std::thread(std::ref(_Xs[i]),
                                  std::ref(*(&first+i)),
                                  Mi));
    ++i;
  }

  TLinearEstimator::operator()(*(&first+i),
                               q + (i < r ? 1 : 0));

  for (auto & th : threads) th.join();

  for (auto const & Y : _Xs) *this += Y;
  return *this;
};
```

Definition of the method operator

Call of `operator()` in the main thread

```
  while (i < _nb_threads-1) {
    _Xs[i].reinit();
    unsigned Mi = q + (i < r ? 1 : 0);
    threads.push_back(std::thread(std::ref(_Xs[i]),
                                  std::ref(*(&first+i)),
                                  Mi));
    ++i;
  }

  TLinearEstimator::operator()(*(&first+i),
                               q + (i < r ? 1 : 0));

  for (auto & th : threads) th.join();

  for (auto const & Y : _Xs) *this += Y;
  return *this;
};
```

Definition of the method operator

Call of the join method

```
  while (i < _nb_threads-1) {
    _Xs[i].reinit();
    unsigned Mi = q + (i < r ? 1 : 0);
    threads.push_back(std::thread(std::ref(_Xs[i]),
                                  std::ref(*(&first+i)),
                                  Mi));
    ++i;
  }

  TLinearEstimator::operator()(*(&first+i),
                               q + (i < r ? 1 : 0));

  for (auto & th : threads) th.join();

  for (auto const & Y : _Xs) *this += Y;
  return *this;
};
```

Definition of the method operator

Assume that `operator+=` is defined for `TLinearEstimator`

```
  while (i < _nb_threads-1) {
    _Xs[i].reinit();
    unsigned Mi = q + (i < r ? 1 : 0);
    threads.push_back(std::thread(std::ref(_Xs[i]),
                                  std::ref(*(&first+i)),
                                  Mi));
    ++i;
  }

  TLinearEstimator::operator()(*(&first+i),
                               q + (i < r ? 1 : 0));

  for (auto & th : threads) th.join();

  for (auto const & Y : _Xs) *this += Y;
  return *this;
};
```

Examples

```
for (unsigned k = 1; k <= hw_threads; ++k) {
  auto estimator = make_multilevel(/* some parameters... */);
  auto par_estimator = make_parallelize(estimator, k);
  auto error = make_rms_error(par_estimator, N, true_value);
  error(gens[0], M_erreur_L2);
  std::cout << k << "\t" << error.time() << std::endl;
}
```

Examples

```
for (unsigned k = 1; k <= hw_threads; ++k) {
  auto estimator = make_multilevel(/* some parameters... */);
  auto par_estimator = make_parallelize(estimator, k);
  auto error = make_rms_error(par_estimator, N, true_value);
  error(gens[0], M_erreur_L2);
  std::cout << k << "\t" << error.time() << std::endl;
}
```

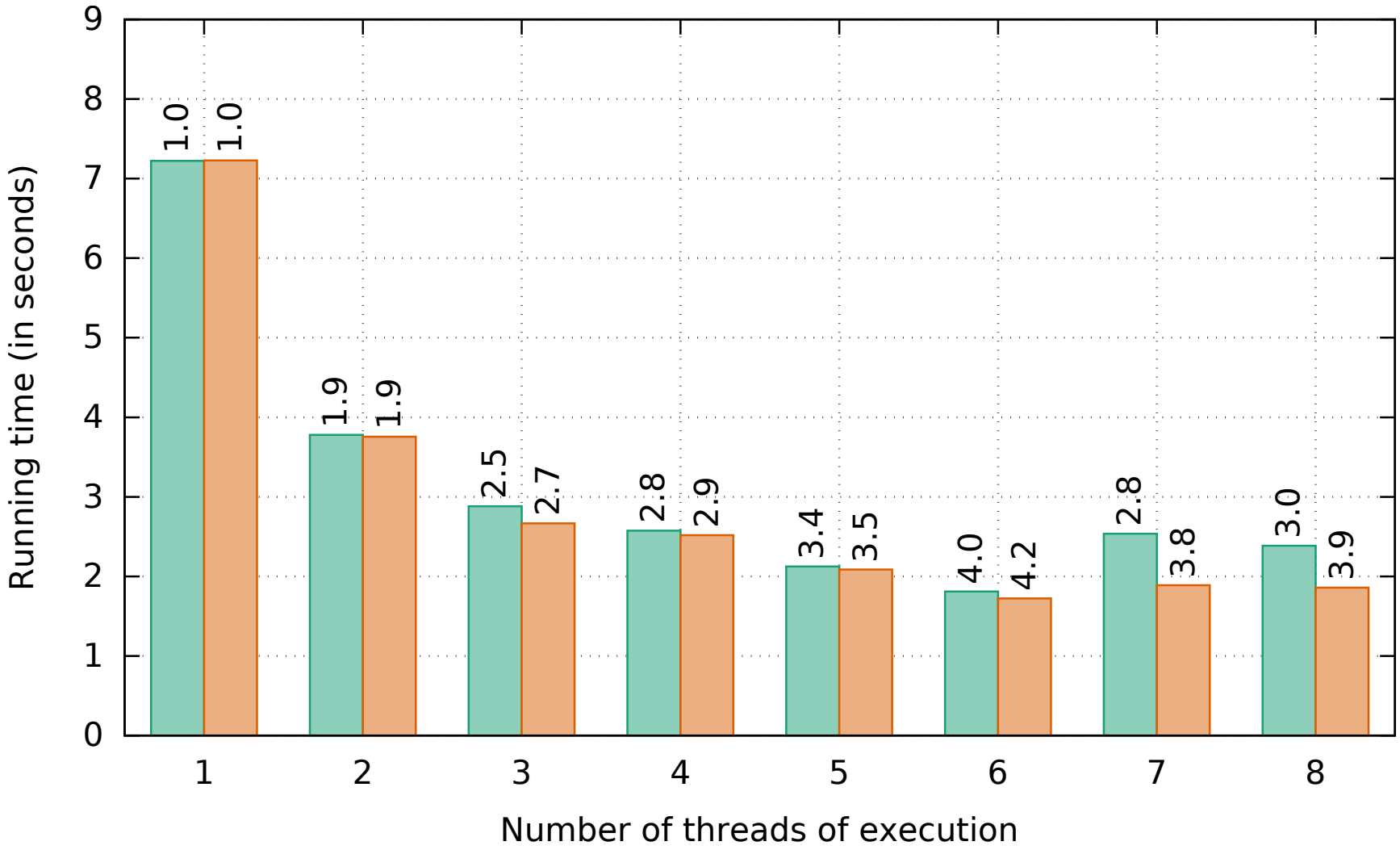
```
for (unsigned k = 1; k <= hw_threads; ++k) {
  auto estimator = make_multilevel(/* some parameters... */);
  auto error = make_rms_error(estimator, N, true_value);
  auto par_error = make_parallelize(error, k);
  par_error(gens[0], M_erreur_L2);
  std::cout << k << "\t" << error.time() << std::endl;
}
```

Examples

```
for (unsigned k = 1; k <= hw_threads; ++k) {
  auto estimator = make_multilevel(/* some parameters... */);
  auto par_estimator = make_parallelize(estimator, k);
  auto error = make_rms_error(par_estimator, N, true_value);
  error(gens[0], M_erreur_L2);
  std::cout << k << "\t" << error.time() << std::endl;
}
```

```
for (unsigned k = 1; k <= hw_threads; ++k) {
  auto estimator = make_multilevel(/* some parameters... */);
  auto error = make_rms_error(estimator, N, true_value);
  auto par_error = make_parallelize(error, k);
  par_error(gens[0], M_erreur_L2);
  std::cout << k << "\t" << error.time() << std::endl;
}
```

Result on Nephtys



Result on Pharaon2