Optimizing Large-Scale ODE Simulations

Mario Mulansky
Institute for Complex Systems, Florence, Italy

![Graph showing speedup for Intel Xeon and AMD Opteron processors with and without SIMD and cache optimization.](image-url)
Optimizing Large-Scale ODE Simulations

Mario Mulansky

Institute for Complex Systems, Florence, Italy

- Performance bottle neck: memory bandwidth

![Graph showing speedup for Intel Xeon and AMD Opteron with and without SIMD and cache optimization.](image-url)
Optimizing Large-Scale ODE Simulations

Mario Mulansky

Institute for Complex Systems, Florence, Italy

- Performance bottle neck: memory bandwidth
- Optimize memory access: bandwidth usage → performance

![Graph showing speedup comparison between Intel Xeon and AMD Opteron with cache optimization and SIMD.]
Optimizing Large-Scale ODE Simulations

Mario Mulansky
Institute for Complex Systems, Florence, Italy

- Performance bottle neck: memory bandwidth
- Optimize memory access: bandwidth usage $\downarrow$ performance
- Utilize SIMD instructions: performance $\uparrow$
Data bandwidth and latency limitation

Modern CPUs: $\sim 3$ GHz, 1 Op/cycle $\rightarrow 3$ GFlops/s
Data bandwidth and latency limitation

Modern CPUs: \( \sim 3 \) GHz, 1 Op/cycle \( \rightarrow \) 3 GFlops/s

Problem: Data transfer from memory to the CPU \( \sim \): 16 GB/s

Example \( \mathbf{x} = \mathbf{a} + \mathbf{b} \): 24 Bytes/Operation

3 GFlops/s \( \leftrightarrow \) 72 GByte/s or 16 GByte/s \( \leftrightarrow \) 0.7 GFlop/s
Data bandwidth and latency limitation

Modern CPUs: \( \sim 3 \text{ GHz}, 1 \text{ Op/cycle} \rightarrow 3 \text{ GFlops/s} \)

Problem: Data transfer from memory to the CPU \( \sim: 16 \text{ GB/s} \)

Example \( \mathbf{x} = \mathbf{a} + \mathbf{b} \): 24 Bytes/Operation

\( 3 \text{ GFlops/s} \leftrightarrow 72 \text{ GByte/s} \text{ or } 16 \text{ GByte/s} \leftrightarrow 0.7 \text{ GFlop/s} \)

Cache

Intel Sandy Bridge

<table>
<thead>
<tr>
<th>Cache</th>
<th>Size</th>
<th>GB/s</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1</td>
<td>32K</td>
<td>350</td>
<td>4</td>
</tr>
<tr>
<td>L2</td>
<td>256K</td>
<td>250</td>
<td>12</td>
</tr>
<tr>
<td>L3</td>
<td>3-6M</td>
<td>100</td>
<td>12-30</td>
</tr>
<tr>
<td>RAM</td>
<td>4-32G</td>
<td>16</td>
<td>20-100</td>
</tr>
</tbody>
</table>
Cache effect: Example

Integrate system of $N$ independent ODEs: $\dot{r}_i = f(r_i, t)$

(Parameter study, Monte-Carlo)
Cache effect: Example

Integrate system of $N$ independent ODEs: $\dot{r}_i = f(r_i, t)$

(Parameter study, Monte-Carlo)

Two approaches:

- time first, $r_1(t) \rightarrow r_1(t + \Delta t) \rightarrow r_1(t + 2\Delta t) \ldots$, then $r_2 \ldots$
- vector first, $r(t) \rightarrow r(t + \Delta t) \rightarrow r(t + 2\Delta t) \ldots$
Cache effect: Example

Integrate system of $N$ independent ODEs: $\dot{r}_i = f(r_i, t)$

(Parameter study, Monte-Carlo)

Two approaches:

- time first, $r_1(t) \rightarrow r_1(t + \Delta t) \rightarrow r_1(t + 2\Delta t) \ldots$, then $r_2 \ldots$
- vector first, $\mathbf{r}(t) \rightarrow \mathbf{r}(t + \Delta t) \rightarrow \mathbf{r}(t + 2\Delta t) \ldots$

![Graph showing Flops/s versus memory size for time first and vector first approaches.](image)
Cache effect: Example

Integrate system of $N$ independent ODEs: $\dot{r}_i = f(r_i, t)$

(Parameter study, Monte-Carlo)

Two approaches:
- time first, $r_1(t) \rightarrow r_1(t + \Delta t) \rightarrow r_1(t + 2\Delta t) \ldots$, then $r_2 \ldots$
- vector first, $\mathbf{r}(t) \rightarrow \mathbf{r}(t + \Delta t) \rightarrow \mathbf{r}(t + 2\Delta t) \ldots$

Flops/s

<table>
<thead>
<tr>
<th>L1</th>
<th>L2</th>
<th>L3</th>
</tr>
</thead>
<tbody>
<tr>
<td>2K</td>
<td>32K</td>
<td>512K</td>
</tr>
<tr>
<td>8M</td>
<td>128M</td>
<td>2G</td>
</tr>
</tbody>
</table>
Simulate chains with nearest neighbor coupling

ODE:

\[ \dot{r}_i = f_i(r, t) = h_i(r_i, t) + g_i(r_i, r_{i-1}, r_{i+1}, t) \]
Simulate chains with nearest neighbor coupling

ODE:

\[
\dot{r}_i = f_i(r, t) = h_i(r_i, t) + g_i(r_i, r_{i-1}, r_{i+1}, t)
\]
Simulate chains with nearest neighbor coupling

ODE:

\[ \dot{r}_i = f_i(r, t) = h_i(r_i, t) + g_i(r_i, r_{i-1}, r_{i+1}, t) \]

Problem: Coupling prevents “time first” approach
Simulate chains with nearest neighbor coupling

ODE:

\[ \dot{r}_i = f_i(r, t) = h_i(r_i, t) + g_i(r_i, r_{i-1}, r_{i+1}, t) \]

Problem: Coupling prevents “time first” approach

“vector first” → slow
Improve Cache Usage: Granularity

Coupling: only “vector first”?
Improve Cache Usage: Granularity

Coupling: only “vector first”? 

Use nearest neighbor coupling

Clusters + overlap comput.
Improve Cache Usage: Granularity

Coupling: only “vector first”?
Use nearest neighbor coupling
Clusters + overlap comput.

\[
\begin{align*}
& r_{g-s} \ldots r_{g-1} \quad r_g \quad r_{g+1} \ldots r_{g-1} \quad r_g \quad \ldots \quad r_{g+s} \\
& r'_g \quad \ldots \quad r'_{g-1} \\
& k^1_g \quad k^1_{g-1} \\
& k^2_g \quad k^2_{g-1} \\
& \ldots \\
& k^s_g \quad \ldots \quad k^s_{g-1} \\
& \text{neglect} \quad \ldots \quad \text{neglect} \\
& \text{final result:} \quad r_g \quad r_{g+1} \quad \ldots \quad r_{g-1}
\end{align*}
\]
Improve Cache Usage: Granularity

Coupling: only “vector first”?  
Use nearest neighbor coupling  
Clusters + overlap comput.  
s iterations for each cluster at once → better cache usage
Improve Cache Usage: Granularity

Coupling: only “vector first”?

Use nearest neighbor coupling

Clusters + overlap comput.

$s$ iterations for each cluster at once → better cache usage

Price: additional overlap computations
Improve Cache Usage: Granularity

Coupling: only “vector first”? Use nearest neighbor coupling

Clusters + overlap comput.

$s$ iterations for each cluster at once → better cache usage

Price: additional overlap computations

Optimal granularity?
Improve Cache Usage: Granularity

Coupling: only “vector first”? 
Use nearest neighbor coupling
Clusters + overlap comput.
$s$ iterations for each cluster at once $\rightarrow$ better cache usage

Price: additional overlap computations

Optimal granularity? Measure!

\[
\begin{align*}
\text{left overlap} & : r_{g-s} \cdots r_g \\
\text{cluster} & : r_g \cdots r_{g+s} \\
\text{right overlap} & : r_g \cdots r_{g+s}
\end{align*}
\]

\[
\begin{align*}
\text{final result:} & \quad r_g \quad r_{g+1} \quad \cdots \quad r_g \quad r_{g-1} \\
\text{neglect} & \quad \text{neglect}
\end{align*}
\]
Performance Results

Coupled Rössler systems, $N = 2^{20} \approx 10^6$ (24 MB)

Intel Xeon E5-2690 @ 3.8GHz
Performance Results

Coupled Rössler systems, $N = 2^{20} \approx 10^6$ (24 MB)

Intel Xeon E5-2690 @ 3.8GHz
Performance Results

Coupled Rössler systems, \( N = 2^{20} \approx 10^6 \) (24 MB)

Intel Xeon E5-2690 @ 3.8GHz
Performance Results

Coupled Rössler systems, \( N = 2^{20} \approx 10^6 \) (24 MB)

Intel Xeon E5-2690 @ 3.8GHz

Bandwidth bound \(\rightarrow\) Flops/s bound
Increase Flops/s: SIMD instructions

SIMD: Single Instruction Multiple Data
Increase Flops/s: SIMD instructions

SIMD: Single Instruction Multiple Data
Increase Flops/s: SIMD instructions

SIMD: Single Instruction Multiple Data

Additional registers and instructions in modern CPUs
Compilers try to use those automatically
Often, explicit SIMD code improve Flops/s significantly
Increase Flops/s: SIMD instructions

SIMD: Single Instruction Multiple Data

Additional registers and instructions in modern CPUs
Compilers try to use those automatically
Often, explicit SIMD code improve Flops/s significantly
Only helpful, if algorithm is Flops bound!
Explicitely use SIMD instructions
Explicitly use SIMD instructions

![Graph showing performance results with SIMD](image)
Explicitly use SIMD instructions

- Performance (steps/s)
  - Standard
  - Cache optimized
  - Cache optimized + SIMD

- MFlops/s
  - Standard
  - Cache optimized
  - Cache optimized + SIMD

- Bandwidth GB/s
  - L2
  - L3
Summary & Conclusions

Granularity → data transfer ↓

BW bound → Flops/s bound

Increase Op/cycle via SIMD
Summary & Conclusions

Granularity → data transfer ↘

BW bound → Flops/s bound

Increase Op/cycle via SIMD

Here: performance gain factor 3

More general: data size > L2 cache size → introduce granularity

Source code: https://github.com/mariomulansky/olsos

C++ with Boost.odeint and Boost.SIMD

∼ 200 lines of code

Thank you for your attention
Summary & Conclusions

Granularity → data transfer ↘

BW bound → Flops/s bound

Increase Op/cycle via SIMD

Here: performance gain factor 3

More general: data size > L2 cache size → introduce granularity
Summary & Conclusions

Granularity → data transfer ↓

BW bound → Flops/s bound

Increase Op/cycle via SIMD

Here: performance gain factor 3

More general: data size > L2 cache size → introduce granularity

Source code: https://github.com/mariomulansky/olsos

C++ with Boost.odeint and Boost.SIMD ~ 200 lines of code
Summary & Conclusions

Granularity $\rightarrow$ data transfer $\downarrow$

BW bound $\rightarrow$ Flops/s bound

Increase Op/cycle via SIMD

Here: performance gain factor 3

More general: data size $>\ L2\ cache\ size \rightarrow$ introduce granularity

Source code: https://github.com/mariomulansky/olsos

C++ with Boost.odeint and Boost.SIMD $\sim 200$ lines of code

Thank you for your attention