Static versus Dynamic Memory Allocation: 
a Comparison for Linear Algebra Kernels

Toufik Baroudi\textsuperscript{1}  \hspace{1cm} Vincent Loechner\textsuperscript{2}  \hspace{1cm} Rachid Seghir\textsuperscript{1}

\textsuperscript{1}University of Batna, Algeria

\textsuperscript{2}University of Strasbourg & INRIA, France

IMPACT 2020
Two years ago [BSL17, TACO]:

- compact data layout for regular sparse matrices
- optimized by Pluto
- our preliminary benchmarks were inconsistent

→ due to matrix allocation mode: as static declared array or as array of pointers to dynamically allocated memory.
We precisely analyze one code: triangular matrix multiplication

- using the performance counters (\#instr., \#mem. access, \#L1-L3 cache misses, \#TLB misses, \#vectorized instr.)

Ran the same tests on the PolyBench linear algebra kernels
Array allocation mode influences performance!

Main factors of performance variation:
- ability of the compiler to detect vectorization
- number of cache misses and memory loads
Array allocation mode influences performance!

Main factors of performance variation:

- ability of the compiler to detect vectorization
- number of cache misses and memory loads

This work is not a manifest for one type of allocation or the other, it is a warning: declaration and allocation of arrays matters!

Comparing various versions of codes using different array allocation modes can get biased
1. Introduction
2. Triangular Matrix Multiplication: Demonstration
3. Triangular Matrix Multiplication: Performance Analysis
4. PolyBench: Performance Analysis
5. Conclusion
Motivation

Triangular Matrix Multiplication

Demo

in completely different conditions than in the paper:
on this laptop (MacOS 10.14, clang/llvm-9.0.0, 4-cores Intel core i7)
Table of Contents

1 Introduction

2 Triangular Matrix Multiplication: Demonstration

3 Triangular Matrix Multiplication: Performance Analysis

4 PolyBench: Performance Analysis

5 Conclusion
Triangular Matrix Multiplication: setup

- Intel platform: dual socket Intel Xeon E5-2650v3 (Haswell-EP)
  2x10 hyperthreaded cores, AVX2 (256 bits)
- AMD platform: dual socket AMD Opteron 6172 (Magny-Cours)
  2x12 cores, SSE (128 bits)
- using pluto-0.11.4 --tile --parallel
- using gcc-7.4.0 -O3 -march=native -fopenmp
- on a regular Linux 4.0.15 (Ubuntu)
- problem size: N=8000
Triangular Matrix Multiplication: execution time

<table>
<thead>
<tr>
<th></th>
<th>Intel C1</th>
<th>AMD C1</th>
<th>Intel C2</th>
<th>AMD C2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orig Stat</td>
<td>166</td>
<td>446</td>
<td>64.8</td>
<td>135.1</td>
</tr>
<tr>
<td>Orig Dyn</td>
<td>209</td>
<td>512</td>
<td>64.1</td>
<td>135.2</td>
</tr>
<tr>
<td>Par Stat</td>
<td>19.8</td>
<td>25</td>
<td>9.7</td>
<td>40.9</td>
</tr>
<tr>
<td>Par Dyn</td>
<td>10.2</td>
<td>20</td>
<td>17.3</td>
<td>34.2</td>
</tr>
</tbody>
</table>

Baroudi, Loechner, Seghir
Static vs. Dynamic Memory Allocation
IMPACT 2020
9 / 25
Triangular Matrix Multiplication: L1-dcache-loads

![Bar chart showing execution time and L1-dcache-ld for different systems and configurations.](chart.png)
Triangular Matrix Multiplication: L1-dcache-misses

Baroudi, Loechner, Seghir
Static vs. Dynamic Memory Allocation
IMPACT 2020 11 / 25
Triangular Matrix Multiplication: L3-dcache-misses

Baroudi, Loechner, Seghir
Static vs. Dynamic Memory Allocation

 execution time (s)

# L3-dc-misses (billions)
Triangular Matrix Multiplication: dTLB-misses

Baroudi, Loechner, Seghir
Static vs. Dynamic Memory Allocation
IMPACT 2020
Triangular Matrix Multiplication: vectorized instructions

unavailable on the AMD but “gcc -fopt-info-vec” seems to confirm the correlation
array allocation mode has a significant impact on the performance of this code
it can have opposite effects on different processors!
factors of influence:
  - number of memory accesses
  - number of cache and TLB misses
  - number of vectorized instructions
other experiments\(^1\) on the Intel platform show that the number of vectorized instructions is a major factor of influence

---
\(^1\)on other triangular matrix kernels: Cholesky, SolveMat, sspfa.
PolyBench: setup

- on the Intel platform
- using `pluto-0.11.4 --tile --parallel`
- using `gcc-7.4.0 -O3 -march=native -fopenmp`
- PolyBench macro `POLYBENCH_STACK ARRAYS`:
  - static version: stack allocated static array
  - dynamic version: multidimensional heap-allocated array
    (not an array of pointers as in the previous experiment)
- problem size:
  - $N=2,000$ for $O(N^3)$ algorithms
  - $N=20,000$ for $O(N^2)$ algorithms
PolyBench: execution time

![Bar graph showing execution time comparison]

**Execution Time (s)**

- **Orig Stat**: Blue bars
- **Orig Dyn**: Red bars
- **Par Stat**: Green bars
- **Par Dyn**: Pink bars

**Software Programs**
- 2mm
- 3mm
- atax
- bicg
- mvt
- trisolv
- lu
- cholesky

**Improvements**
- 3.8x improvement
- +20% increase
PolyBench: vectorized instructions

Baroudi, Loechner, Seghir
Static vs. Dynamic Memory Allocation
IMPACT 2020
PolyBench: L1 cache misses

![Bar chart showing L1 cache misses for various benchmarks under different memory allocation strategies.](image)

- **Orig Stat**: Original static memory allocation
- **Orig Dyn**: Original dynamic memory allocation
- **Par Stat**: Parallel static memory allocation
- **Par Dyn**: Parallel dynamic memory allocation

The charts display execution time (in seconds) and the number of L1 cache misses (in millions) for different benchmarks:

- **2mm**
- **3mm**
- **atax**
- **bicg**
- **mvt**
- **trisolv**
- **lu**
- **cholesky**

The y-axis represents execution time in seconds on a logarithmic scale, ranging from $10^{-1}$ to $10^3$. The x-axis lists the benchmarks.

The bottom chart shows the number of L1 cache misses in millions, with a similar logarithmic scale ranging from $10^1$ to $10^5$. The benchmarks and memory allocation strategies are the same as above.

These charts illustrate the impact of static vs. dynamic memory allocation on performance and cache efficiency for various PolyBench benchmarks.
PolyBench: memory loads

![Diagram showing memory loads for different benchmarks and allocator strategies.](image)

### L1-dcache-loads (billions)

- **Orig Stat**: Blue bars
- **Orig Dyn**: Red bars
- **Par Stat**: Green bars
- **Par Dyn**: Pink bars

### Execution time (s)

- **Orig Stat**: Blue bars
- **Orig Dyn**: Red bars
- **Par Stat**: Green bars
- **Par Dyn**: Pink bars

### Benchmarks
- 2mm
- 3mm
- atax
- bicg
- mvt
- trisolv
- lu
- cholesky

### Execution Times

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Orig Stat</th>
<th>Orig Dyn</th>
<th>Par Stat</th>
<th>Par Dyn</th>
</tr>
</thead>
<tbody>
<tr>
<td>2mm</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3mm</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>atax</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bicg</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mvt</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>trisolv</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lu</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cholesky</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### L1-dcache-loads (billions)

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Orig Stat</th>
<th>Orig Dyn</th>
<th>Par Stat</th>
<th>Par Dyn</th>
</tr>
</thead>
<tbody>
<tr>
<td>2mm</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3mm</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>atax</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bicg</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mvt</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>trisolv</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lu</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cholesky</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
PolyBench: memory loads

exec. time (s)

# L1-dcache-loads (billions)

Baroudi, Loechner, Seghir
Static vs. Dynamic Memory Allocation
IMPACT 2020
Analysis

- factors of influence:
  - number of vectorized instructions
  - number of cache misses
  - number of memory accesses?
  - ?

- why are there more/less memory accesses when allocating arrays statically or dynamically?
  → we suspect that the varying pressure on register allocation changes the compiler’s decision on data reuse (bicg)
Table of Contents

1 Introduction

2 Triangular Matrix Multiplication: Demonstration

3 Triangular Matrix Multiplication: Performance Analysis

4 PolyBench: Performance Analysis

5 Conclusion
Array allocation has a **significant impact** on performance in many cases.

It can be alternatively in favor of static or dynamic allocation and it can even flip on different architectures!

Be careful when you compare codes using different allocations (e.g. when working on data layout transformations): this side effect **could bias your measurements**!