Dynamic Load Balancing of Loops in OpenMP

Florina M. Ciorba

SC19 Denver, 19.11.2019
11:15 am
OpenMP Booth #1737

University of Basel, Switzerland
Outline

✧ Load imbalance
✧ OpenMP loops and their scheduling
✧ **Dynamic Loop Self-Scheduling (DLS) for Load Balancing in GNU OpenMP**
✧ DLS for Load Balancing in **LLVM OpenMP**
✧ Take away messages
Acknowledgments and Collaborators

MLS: Multilevel Scheduling in Large Scale High Performance Computers

F. M. Ciorba
J. H. Müller Korndörfer
A. Mohammed
A. Yilmaz
P. Buder

C. Iwainsky
F. Kasielke
R. Tschüter
M. Velten

I. Banicescu
M. Klemm

J. Doerfert
H. Finkel

V. Kale
Increased Software Parallelism

✧ Computationally-intensive applications
  ✧ Large in time and/or space
  ✧ Data parallel
  ✧ Irregular in time and/or space: problem, algorithm, implementation
Increased Software Parallelism

✧ Computationally-intensive applications
  ✧ Large in time and/or space
  ✧ Data parallel
  ✧ Irregular in time and/or space: problem, algorithm, implementation

✧ Many applications often spend a significant amount of time executing loops
Increased Hardware Parallelism

✧ Increasingly complex and large (node and cores/node counts)
✧ Highly heterogeneous (CPUs, GPUs, FPGAs, TPUs, VPs, etc.)
✧ Exhibit massive and diverse (hardware) parallelism
**Increased Hardware Parallelism**

- Increasingly complex and large (node and cores/node counts)
- Highly heterogeneous (CPUs, GPUs, FPGAs, TPUs, VPUs, etc.)
- Exhibit massive and diverse (hardware) parallelism

[Images of data visualizations showing the progression from 1 core in Top 500 Nov 2001 to 6-260 cores in Top 500 Nov 2018]
Increased Hardware Parallelism

- Increasingly complex and large (node and cores/node counts)
- Highly heterogeneous (CPUs, GPUs, FPGAs, TPUs, VPIs, etc.)
- Exhibit massive and diverse (hardware) parallelism

How does the increased SW parallelism execute on large parallel machines?
Load Imbalance

- Timeline view generated with Vampir (www.vampir.eu)
- SPHYNX, a smoothed particle hydrodynamics (SPH) code used in astrophysics

There are idle processors and there is work ready to be performed that no processor has started
Load Imbalance

- Timeline view generated with Vampir (www.vampir.eu)
- SPHYNX, a smoothed particle hydrodynamics (SPH) code used in astrophysics

https://astro.physik.unibas.ch/people/ruben-cabezon/sphynx.html
Load Imbalance

- Timeline view generated with Vampir (www.vampir.eu)
- SPHYNX, a smoothed particle hydrodynamics (SPH) code used in astrophysics


Load imbalance at thread level (OpenMP loop)

How to address load imbalance in OpenMP parallel loops?

https://astro.physik.unibas.ch/people/ruben-cabezon/sphynx.html
**Scheduling of OpenMP Loops**

✧ **Scheduling**: performance critical aspect of executing loops

✧ **Loops**: an important part of many science, engineering, and industry OpenMP programs

✧ Complemented, not overshadowed, by the introduction of *explicit tasks* in OpenMP

✧ A *central parallelization approach* based on *work sharing* of iterations into chunks

✧ **Load imbalance**: uneven progress during execution (idle processors while work available)

```c
#pragma omp schedule(static,chunk)
```

Unused CPU time => Load imbalance

```c
#pragma omp schedule(???)
```

Unused CPU time => Load imbalance
Loop Scheduling Options in OpenMP

OpenMP standard `schedule()`

- **static,chunk**: predetermined allocation order, offset by thread ID (no self-scheduling)
- **dynamic,1**: pure self-scheduling SS [Lusk, Overbeek ‘83]
- **dynamic,chunk**: chunk self-scheduling CSS [Kruskal, Weiss ‘85] with exact chunk size = “chunk”
- **guided**: guided self-scheduling GSS [Polychronopoulos, Kuck ‘87]
- **guided,chunk**: GSS with minimum chunk size = “chunk”
- **auto**: implementation determines schedule; no “chunk” support
Loop Scheduling Options in OpenMP

OpenMP standard `schedule()`

- **static,chunk**: predetermined allocation order, offset by thread ID (no self-scheduling)
- **dynamic,1**: pure self-scheduling SS [Lusk, Overbeek ‘83]
- **dynamic,chunk**: chunk self-scheduling CSS [Kruskal, Weiss ‘85] with exact chunk size = “chunk”
- **guided**: guided self-scheduling GSS [Polychronopoulos, Kuck ‘87]
- **guided,chunk**: GSS with minimum chunk size = “chunk”
- **auto**: implementation determines schedule; no “chunk” support

Are these schedules good enough to efficiently exploit HW parallelism in 2019+?
Loop Scheduling Options in OpenMP

OpenMP standard `schedule()`

- **static,chunk**: predetermined allocation order, offset by thread ID (no self-scheduling)
- **dynamic,1**: pure self-scheduling SS [Lusk, Overbeek ‘83]
- **dynamic,chunk**: chunk self-scheduling CSS [Kruskal, Weiss ‘85] with exact chunk size = “chunk”
- **guided**: guided self-scheduling GSS [Polychronopoulos, Kuck ‘87]
- **guided,chunk**: GSS with minimum chunk size = “chunk”
- **auto**: implementation determines schedule; no “chunk” support

**Are these schedules good enough to efficiently exploit HW parallelism in 2019+?**

**Are these schedules sufficient to cover the characteristics of all apps and systems?**
Loop Scheduling Options in OpenMP

OpenMP standard `schedule()`

- **static,chunk**: predetermined allocation order, offset by thread ID (no self-scheduling)
- **dynamic,1**: pure self-scheduling SS [Lusk, Overbeek ‘83]
- **dynamic,chunk**: chunk self-scheduling CSS [Kruskal, Weiss ‘85] with exact chunk size = “chunk”
- **guided**: guided self-scheduling GSS [Polychronopoulos, Kuck ‘87]
- **guided,chunk**: GSS with minimum chunk size = “chunk”
- **auto**: implementation determines schedule; no “chunk” support

Are these schedules **good enough** to efficiently exploit HW parallelism in 2019+?

Are there better schedules **not yet** in OpenMP?

Are these schedules **sufficient** to cover the characteristics of all apps and systems?
Loop Scheduling Options in OpenMP

OpenMP standard `schedule()`

- **static,chunk**: predetermined allocation order, offset by thread ID (no self-scheduling)
- **dynamic,1**: pure self-scheduling SS [Lusk, Overbeek '83]
- **dynamic,chunk**: chunk self-scheduling CSS [Kruskal, Weiss '85] with exact chunk size = “chunk”
- **guided**: guided self-scheduling GSS [Polychronopoulos, Kuck '87]
- **guided,chunk**: GSS with minimum chunk size = “chunk”
- **auto**: implementation determines schedule; no “chunk” support

Are these schedules **good enough** to efficiently exploit HW parallelism in 2019+?

Yes!

Are there better schedules **not yet** in OpenMP?

Are these schedules **sufficient** to cover the characteristics of all apps and systems?

Yes!
Dynamic Load Balancing of OpenMP Loops*

- Timeline view generated with Vampir (www.vampir.eu)
  - SPHYNX, a smoothed particle hydrodynamics (SPH) code used in astrophysics


dynamic_load_balancing

What Causes Load Imbalance in the Execution of OpenMP Loops?

✧ Computationally-intensive and **irregular** loops
  ✧ Non-uniform data distribution
  ✧ Differences in complexity per iteration
  ✧ Boundary phenomena, convergence, conditions, etc.
  
  \[ \text{variable iteration execution times} \]

✧ **System**-induced variability
  ✧ Differences in hardware (PE variability)
  ✧ NUMA, OS jitter, resource sharing, etc.

✧ Synchronization (allocation **delays**) 
✧ Process management (**scheduling overhead**) 
✧ Communication (**locality**)
What Causes Load Imbalance in the Execution of OpenMP Loops?

- Computationally-intensive and irregular loops
  - Non-uniform data distribution
  - Differences in complexity per iteration
  - Boundary phenomena, convergence

- System-induced variability
  - Differences in hardware (PE variability)
  - NUMA, OS jitter, resource sharing, etc.

- Synchronization (allocation delays)
- Process management (scheduling overhead)
- Communication (locality)

The impact of system-induced variability on load imbalance is often neglected in loop scheduling research, particularly by available OpenMP scheduling choices.
Dynamic Loop Self-Scheduling (DLS)

Self-scheduling [Smith, 1981] [Tang, Yew, 1986]
✧ Among the first dynamic parallel-loop scheduling as optimized implementation of List Scheduling [(2-1/m)-approximate of optimal preemptive schedule]
✧ Allow each process to acquire the next computational step dynamically when it finishes the previous one.
✧ Most attractive benefit
  ✧ There is no need to worry about poor process utilization resulting from an unsatisfactory a priori loop schedule
✧ Reduces the overhead of dynamic scheduling associated with system calls to the OS
  ✧ Rather than issuing an OS system call for scheduling, processors can schedule themselves by fetch-and-adding a shared variable to get the loop indices of a chunk of iterations
  ✧ Applies to nested loops as well
✧ Implemented in OpenMP as dynamic,1: pure self-scheduling SS [Lusk, Overbeek ‘83]

Dominant heuristic in online nonclairvoyance scheduling

Selected Self

C_{\text{max}}: (2 - \frac{\text{optimal preemptive schedule}}{m}) - \frac{1}{m}

Among the first dynamic parallel loop scheduling as optimized implementation of List scheduling (Graham, 1966)

1966

1981

1986

1989

1991

1995

1997

2001

2006

1983

1986

1989

1993

1999

2008

List scheduling

Dominant heuristic in online nonclairvoyance scheduling

P | C_{\text{max}} \geq (2-1/m)-approximate of optimum (optimal preemptive schedule)

Jobs executed from a pre-specified list whenever a machine becomes idle, assuming all precedence constraints are satisfied

FSC | CSS, ECSS

(Kruskal, Weiss)

Fixed-size chunking or uniform-sized chunking. Found as "static" in OpenMP. Enhanced CSS assigns dependent iterations to the same processor

1985

1987

1990

1992

1994

1996

2000

Dynamic Load Balancing of Loops...
Choice of the scheduling strategy and its potential to mitigate the imbalance significantly affects the performance of the parallel loop.
Loop Scheduling in Compilers and Runtime Systems

1987
- Implememts FSC
  - (Butler et al.)
- Implements AGSS
  - (Eager, Zahorjan)
- Adaptive Guided Self-Scheduling
  - Proposed a wrapped assignment of iterations to rectify a shortcoming of GSS

1988
- TUS
  - (Blumofe et al.)
- Distributed Scheduling System
  - (Nagel)

1989
- ASC
  - (Polychronopoulos)
- Auto-scheduling Compiler
  - Compiler generates drive code as part of object code. Extends runtime libs.

1990
- Cilk
  - (Blumofe et al.)

1991
- Adaptive LB
  - (Ioannidis, Dwarakadas)

1993
- Distributed Cooperative Scheduling System
  - Job scheduler, Task scheduler, Thread scheduler (CS, CSS, GSS, FSC)

1996
- Fixed block SS
  - (Durand et al.)
- Impact of memory contention on dynamic scheduling on NUMA multiprocessors for fixed block SS (CSS) and for decreasing block SS

1998
- DLS API vs. OpenMP
  - (Govindaswamy)
- Parallel Competitive Runtime Environment for Multicore Applications

2000
- IPLS
  - (Fann et al.)

2004
- Adaptive OpenMP
  - Implement AFS, TSS

2005
- Runtime Empirical Selection
  - (Zhang, Voss)

2008
- OpenMP Task Scheduling
  - (Durand et al.)

2011
- autopin
  - (Klug et al.)
- Automated Optimization of Thread-to-Core Pinning on Multicore Systems

2015
- OpenMP Interface to define own user schedule

2017
- DLS+LibGomp
  - (Buder)
- Evaluation and Analysis of Dynamic Loop Scheduling in OpenMP

2019
- DLS+LibGomp
  - (Müller Kormödörfer et al.)
- Implementation, evaluation, analysis of self scheduling in OpenMP

UDS
- (Kale et al.)
- Prototype proposal for User-defined Scheduling Prototype in the OpenMP standard

Dynamic Load Balancing of Loops in OpenMP | F. Ciorba
Enhancing Loop Scheduling in OpenMP

**Challenge:** Enabling user selection of new scheduling strategies
- OpenMP allows only three predefined scheduling strategies
- Specific scheduling ‘locked in’ at compile-time
- **Defer scheduling choice to execution time**
  - #pragma omp for schedule(runtime)
  - Environment variable OMP_SCHEDULE used to choose schedule
- Intel, LLVM, and GNU OpenMP runtimes
  - Evaluate environment variables in the RUNTIME LIBRARY
  - Implement schedule() via functions from the RUNTIME LIBRARY

No compiler modification needed
Suffices to augment the runtime library
More Schedules in the GNU (libgomp) OpenMP runtime library

Selected self-scheduling (not in the OpenMP standard) newly added to the GNU OpenMP runtime library

- **fac2**: practical factoring FAC, 1990-1992
  - Unknown mean and stdev of iteration times
- **tss**: trapezoid self-scheduling TSS, 1993
  - Collapses to static,chunk when first and last chunk equal #iterations/#cores
- **wf2**: practical weighted factoring WF, 1996
  - Unknown mean and stdev of iteration times
- **rand**: random self-scheduling RAND
  - Random chunk ∈ [#iterations/100×#cores, #iterations/2×#cores], min ≥ 1, max ≥ min+1

---

More Schedules in the GNU (libgomp) OpenMP runtime library
iWOMP | Barcelona | 2018

Selected self-scheduling (not in the OpenMP standard) newly added to the GNU OpenMP runtime library
✧ fac2: practical factoring FAC, 1990-1992
✧ tss: trapezoid self-scheduling TSS, 1993
✧ wf2: practical weighted factoring WF, 1996
✧ rand: random self-scheduling RAND

✧ Usage via schedule(runtime) and OMP_SCHEDULE
✧ Implementation into GNU open source OpenMP runtime LaPeSD-libGOMP https://github.com/lapesd/libgomp

✧ Note: Even though the GNU implementation of OpenMP is used here, the scheduling mechanisms are independent from the actual compilation of OpenMP constructs and their separate implementation in a separate runtime library.
More Schedules in the GNU (libgomp) OpenMP runtime library

Systemic Variability

iWOMP | Barcelona | 2018

<table>
<thead>
<tr>
<th>CORES</th>
<th>CORE WEIGHTS</th>
<th>SOCKET 0</th>
<th>SOCKET 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIN1</td>
<td></td>
<td>T0 T1 T2 T3 T4 T5 T6 T7 T8 T9 T10 T11 T12 T13 T14 T15 T16 T17 T18 T19</td>
<td>T0 T1 T2 T3 T4 T5 T6 T7 T8 T9 T10 T11 T12 T13 T14 T15 T16 T17 T18 T19</td>
</tr>
<tr>
<td>PIN2</td>
<td></td>
<td>T0 T1 T2 T3 T4 T5 T6 T7 × ×</td>
<td>T0 T1 T2 T3 T4 T5 T6 T7 T9 T10 T11 T12 T13 T14 T15 T16 T17 ×</td>
</tr>
<tr>
<td>PIN3</td>
<td></td>
<td>T0 T1 T2 T3 T4 T5 T6 × × ×</td>
<td>T0 T1 T2 T3 T4 T5 T6 × × ×</td>
</tr>
<tr>
<td>PIN4</td>
<td></td>
<td>T0 T1 T2 T3 T4 T5 T6 T7 T8 T9 T10 × × × × × × ×</td>
<td>T0 T1 T2 T3 T4 T5 T6 T7 T8 T9 T10 × × × × × × ×</td>
</tr>
<tr>
<td>PIN5</td>
<td></td>
<td>T0 T1 T2 T3 T4 T5 T6 T7 T8 T9 × × × × × × ×</td>
<td>T0 T1 T2 T3 T4 T5 T6 T7 T8 T9 × × × × × × ×</td>
</tr>
</tbody>
</table>

Does a schedule **benefit** a parallel loop?
Can it handle **HW heterogeneity**?
More Schedules in the GNU (libgomp) OpenMP runtime library

Experiments

iWOMP | Barcelona | 2018

✧ Executions with 20 threads using
  ✧ Originally: no schedule clause
  ✧ Existing: STATIC, SS(Dynamic, 1), GSS(Guided)
  ✧ Newly added: FAC2, TSS, WF, RAND
More Schedules in the GNU (libgomp) OpenMP runtime library

Experiments

iWOMP | Barcelona | 2018

- Executions with 20 threads using
  - Originally: no schedule clause
  - Existing: STATIC, SS(Dynamic, 1), GSS(Guided)
  - Newly added: FAC2, TSS, WF, RAND

- System-induced variability
  - Five pinning strategies
  - Intel Xeon CPU E5-2640 v4, 2 sockets
More Schedules in the GNU (libgomp) OpenMP runtime library

Experiments

iWOMP | Barcelona | 2018

- Executions with 20 threads using
  - Originally: *no schedule clause*
  - Existing: STATIC, SS(Dynamic, 1), GSS(Guided)
  - Newly added: FAC2, TSS, WF, RAND

- System-induced variability
  - Five pinning strategies
  - Intel Xeon CPU E5-2640 v4, 2 sockets

- Parallel execution time statistics (median) of 20 repetitions of each (benchmark × schedule × pinning) experiment
More Schedules in the GNU (libgomp) OpenMP runtime library
Experiments
iWOMP | Barcelona | 2018

Adjoint Convolution

<table>
<thead>
<tr>
<th>Schedule Clauses</th>
<th>STATIC</th>
<th>GSS</th>
<th>TSS</th>
<th>FAC2</th>
<th>WF2</th>
<th>RAND</th>
<th>SS</th>
</tr>
</thead>
<tbody>
<tr>
<td>#Iterations</td>
<td>10^6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C.O.V.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exec. Time</td>
<td>57 %</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

max

min
More Schedules in the GNU (libgomp) OpenMP runtime library Experiments

iWOMP | Barcelona | 2018

![Graph showing performance metrics for different schedule clauses.]

- **Adjoint Convolution**
- **Least performing (large first chunk)**
- **Best performing**

<table>
<thead>
<tr>
<th>Schedule</th>
<th>#Iterations</th>
<th>C.O.V.</th>
<th>Exec. Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATIC</td>
<td>$10^6$</td>
<td></td>
<td>57%</td>
</tr>
<tr>
<td>GSS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TSS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FAC2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WF2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RAND</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SS</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Least and most sensitive to pinning**

Dynamic Load Balancing of Loops in OpenMP | F. Ciorba
More Schedules in the GNU (libgomp) OpenMP runtime library
Experiments
iWOMP | Barcelona | 2018

OmpSCR C_MD

<table>
<thead>
<tr>
<th>Schedule Clauses</th>
<th>Pin1</th>
<th>Pin2</th>
<th>Pin3</th>
<th>Pin4</th>
<th>Pin5</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATIC</td>
<td>120</td>
<td>140</td>
<td>160</td>
<td>180</td>
<td>200</td>
</tr>
<tr>
<td>GSS</td>
<td>110</td>
<td>130</td>
<td>150</td>
<td>170</td>
<td>190</td>
</tr>
<tr>
<td>TSS</td>
<td>100</td>
<td>120</td>
<td>140</td>
<td>160</td>
<td>180</td>
</tr>
<tr>
<td>FAC2</td>
<td>90</td>
<td>110</td>
<td>130</td>
<td>150</td>
<td>170</td>
</tr>
<tr>
<td>WF2</td>
<td>80</td>
<td>100</td>
<td>120</td>
<td>140</td>
<td>160</td>
</tr>
<tr>
<td>RAND</td>
<td>70</td>
<td>90</td>
<td>110</td>
<td>130</td>
<td>150</td>
</tr>
<tr>
<td>SS</td>
<td>60</td>
<td>80</td>
<td>100</td>
<td>120</td>
<td>140</td>
</tr>
</tbody>
</table>

#Iterations: $16 \times 10^3$
C.O.V.: 57%
More Schedules in the GNU (libgomp) OpenMP runtime library

Experiments
iWOMP | P

<table>
<thead>
<tr>
<th>Schedule</th>
<th>#Iterations</th>
<th>C.O.V.</th>
<th>Exec. Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATIC</td>
<td>16×10³</td>
<td>57 %</td>
<td></td>
</tr>
</tbody>
</table>

Least performing
Best performing (close to GSS, TSS, WF2)

Most sensitive to pinning (initial calibration, few iterations)
More Schedules in the GNU (libgomp) OpenMP runtime library
iwOMP | Barcelona | 2018

✧ Additional scheduling choices provide benefit over existing schedules

✧ When application and system parallelism is regular, STATIC is sufficient

✧ The newly implemented DLS are immediately usable by existing programs using our non-standard prototype implementation* via schedule(runtime)
  ✧ Numerous OpenMP production codes in active use
  ✧ Numerous multi/manycore platforms available

* https://bitbucket.org/PatrickABuder/libgomp/src/master/
More Schedules in the LLVM (libomp) OpenMP runtime library

Modified files for the implementation of additional scheduling techniques to the LLVM OpenMP runtime library

- Libomp uses the `init()`, `next()`, and `finish()` functions to perform the scheduling of iterations from a loop onto threads
- The added loop scheduling techniques are located in the `kmp_dispatch.cpp`
- The remaining files initialize required environment variables and make the OpenMP runtime library aware of the eleven newly introduced self-scheduling techniques
More Schedules in the LLVM (libomp) OpenMP runtime library

ISPDC | Amsterdam | 2019

Selected self-scheduling (not in the OpenMP standard) newly added to the LLVM OpenMP runtime library

- **fac**: factoring, 1990-1992
- **tss**: trapezoid self-scheduling **TSS**, 1993

---

More Schedules in the LLVM (libomp) OpenMP runtime library
ISPDC | Amsterdam | 2019

Selected self-scheduling (not in the OpenMP standard) newly added to the LLVM OpenMP runtime library
✧ fac: factoring, 1990-1992
✧ tss: trapezoid self-scheduling TSS, 1993

✧ Usage via `schedule(runtime)` and `OMP_SCHEDULE`

✧ **Note:** Even though the LLVM implementation of OpenMP is used here, the scheduling mechanisms are independent from the actual compilation of OpenMP constructs and their separate implementation in a separate runtime library.
More Schedules in the **LLVM (libomp)** OpenMP runtime library

**Experiments**

ISPDC | Amsterdam | 2019

✧ Executions with 24 threads using
  ✧ Originally: *no schedule* clause
  ✧ Existing: STATIC, SS(Dynamic, 1), GSS(Guided)
  ✧ Newly added: FAC, TSS
More Schedules in the LLVM (libomp) OpenMP runtime library Experiments
ISPDC | Amsterdam | 2019

✧ Executions with 24 threads using
  ✧ Originally: no schedule clause
  ✧ Existing: STATIC, SS(Dynamic, 1), GSS(Guided)
  ✧ Newly added: FAC, TSS

✧ System-induced variability
  ✧ 1 pinning strategy (1 thread per core)
  ✧ Intel Xeon E5-2680 v3, 2 sockets
More Schedules in the LLVM (libomp) OpenMP runtime library
Experiments
ISPDC | Amsterdam | 2019

✧ Executions with 24 threads using
  ✧ Originally: *no schedule* clause
  ✧ Existing: STATIC, SS(Dynamic, 1), GSS(Guided)
  ✧ Newly added: FAC, TSS

✧ System-induced variability
  ✧ 1 pinning strategy (1 thread per core)
  ✧ Intel Xeon E5-2680 v3, 2 sockets

✧ Scheduling overhead measurements with EPCC benchmarks (*100 repetitions*)
✧ Parallel execution time statistics (average) of *10 repetitions* of each (benchmark × schedule × pinning) experiment
More Schedules in the LLVM (libomp) OpenMP runtime library Experiments
ISPDC | Amsterdam | 2019

Scheduling overhead measurements using EPCC benchmark
More Schedules in the LLVM (libomp) OpenMP runtime library
Experiments
ISPDC | Amsterdam | 2019

Scheduling overhead measurements using EPCC benchmark

![Graph showing scheduling overhead measurements](image)

- **STATIC** (static)
- **SS** (= guided)
- **TSS**
- **RANDOM**

Legend:
- Standard OpenMP scheduling
- Newly implemented in LLVM
- Already implemented in LLVM
More Schedules in the **LLVM (libomp)** OpenMP runtime library

**Experiments**

ISPDC | Amsterdam | 2019

✧ **Static** performs *poorly*
✧ **Dynamic** performs *best*
✧ **Application characteristics**
  ✧ Comparatively low number of iterations (500\(^2\), 750\(^2\), 1000\(^2\))
  ✧ Inherent work imbalance (lower triangular)

![Adjoint Convolution](image)
More Schedules in the LLVM (libomp) OpenMP runtime library

Experiments

ISPDC | Amsterdam | 2019

- **Dynamic** performs *poorly*
- **Static** performs *best*

Application characteristics:
- Molecular dynamics code (RODINIA suite)
- Less imbalance compared with Adjoint Convolution
More Schedules in the LLVM (libomp) OpenMP runtime library
ISPDC | Amsterdam | 2019

✧ Template for extending OpenMP loop scheduling
  ✧ Implementation of factoring in the LLVM OpenMP runtime
  ✧ Strong similarities in the OpenMP implementations of GNU and LLVM (Intel)
More Schedules in the LLVM (libomp) OpenMP runtime library (extension)

SC Research Poster | Denver | 2019 [Poster 109]

Selected **self-scheduling** and **adaptive self-scheduling** (not in the OpenMP standard) newly added to the LLVM OpenMP runtime library

- **fsc**: fixed size chunking, 1985
- **fac**: factoring, 1990-1992
- **fac2**: practical **factoring FAC**, 1990-1992
- **tap**: taper, 1992
- **wf**: weighted factoring, 1996
- **bold**: 1997
- **awf_(b, c, d, and e)**: **adaptive** weighted factoring and variants, 1999-2000
- **af**: **adaptive** factoring, 2000

Improvements to the **fac**, **fac2**, and **af** techniques are denoted by the suffix **_a**, 2019

---

More Schedules in the **LLVM (libomp)** OpenMP runtime library (extension)

SC Research Poster | Denver | 2019 [Poster 109]

Selected **self-scheduling** and **adaptive self-scheduling** (not in the OpenMP standard) newly added to the **LLVM** OpenMP runtime library

- **fsc**: fixed size chunking, 1985
- **fac**: factoring, 1990-1992
- **fac2**: practical factoring **FAC**, 1990-1992
- **tap**: taper, 1992
- **wf**: weighted factoring, 1996
- **bold**: 1997
- **awf_(b, c, d, and e)**: adaptive weighted factoring and variants, 1999-2000
- **af**: adaptive factoring, 2000
- Improvements to the **fac**, **fac2**, and **af** techniques are denoted by the suffix _a, 2019

- **Usage via schedule(runtime) and OMP_SCHEDULE**
More Schedules in the **LLVM (libomp)** OpenMP runtime library

*Systemic Variability*

**SC Research Poster | Denver | 2019 [Poster 109]**

Does a schedule *benefit* a parallel loop?  
Can it handle *HW heterogeneity*?

---

**PU weights:**  
1 0.5 0.33 0.25

---

PU: Processing unit  
T: OpenMP thread

---

Dynamic Load Balancing of Loops in OpenMP | F. Ciorba
More Schedules in the LLVM (libomp) OpenMP runtime library

Experiments

SC Research Poster | Denver | 2019 [Poster 109]

✧ Executions with 20 and 64 threads using
  ✧ Originally: no schedule clause
  ✧ Existing: STATIC, SS(Dynamic, 1), SS(Dynamic, 64), GSS(Guided), TSS(Trapezoidal)
  ✧ Newly added: FSC, FAC, FAC2, TAP, WF, BOLD, awf_(b, c, d, and e), af,
    Improvements to the fac, fac2, and af techniques denoted by the suffix _a
More Schedules in the LLVM (libomp) OpenMP runtime library Experiments

SC Research Poster | Denver | 2019 [Poster 109]

✧ Executions with 20 and 64 threads using
  ✧ Originally: no schedule clause
  ✧ Existing: STATIC, SS(Dynamic, 1), SS(Dynamic, 64), GSS(Guided), TSS(Trapezoidal)
  ✧ Newly added: FSC, FAC, FAC2, TAP, WF, BOLD, awf_(b, c, d, and e), af,
    Improvements to the fac, fac2, and af techniques denoted by the suffix _a

✧ System-induced variability
  ✧ Two pinning strategies
  ✧ Intel Xeon CPU E5-2640 v4, 2 sockets
  ✧ Intel Xeon Phi CPU 7210 (KNL), 1 socket
More Schedules in the LLVM (libomp) OpenMP runtime library

Experiments

SC Research Poster | Denver | 2019 [Poster 109]

✧ Executions with 20 and 64 threads using
  ✧ Originally: no schedule clause
  ✧ Existing: STATIC, SS(Dynamic, 1), SS(Dynamic, 64), GSS(Guided), TSS(Trapezoidal)
  ✧ Newly added: FSC, FAC, FAC2, TAP, WF, BOLD, awf_(b, c, d, and e), af,
    Improvements to the fac, fac2, and af techniques denoted by the suffix _a

✧ System-induced variability
  ✧ Two pinning strategies
  ✧ Intel Xeon CPU E5-2640 v4, 2 sockets
  ✧ Intel Xeon Phi CPU 7210 (KNL), 1 socket

✧ Parallel execution time statistics (average) of 20 repetitions of each (benchmark × schedule × pinning) experiment
More Schedules in the **LLVM (libomp)** OpenMP runtime library Experiments

SC Research Poster | Denver | 2019  [Poster 109]

For more details and discussion come visit our SC19 poster 109: A Runtime Approach for Dynamic Load Balancing of OpenMP Parallel Loops in LLVM

**NAS 3.4 MG class C - KNL - PIN 1**

**SPECOMP 2012 MD size REF - XEON - PIN 1**
More Schedules in the LLVM (libomp) OpenMP runtime library Experiments

SC Research Poster | Denver | 2019 [Poster 109]

CORAL2 Quicksilver Hybrid MPI+OpenMP (4 XEON nodes)

10M particles - PIN 2

NAS 3.4 MG class C - XEON - PIN 2

For more details and discussion come visit our SC19 poster 109: A Runtime Approach for Dynamic Load Balancing of OpenMP Parallel Loops in LLVM
Take Away Messages

✧ Showed the effectiveness of state-of-the-art loop self-scheduling to address load imbalance due to application and system
✧ Existing OpenMP schedules (static, guided, dynamic) are good for corner cases
Take Away Messages

✧ Showed the effectiveness of state-of-the-art loop self-scheduling to address load imbalance due to application and system
✧ Existing OpenMP schedules (static, guided, dynamic) are good for corner cases
✧ More schedules are needed to add flexibility in the OpenMP runtime(s) against load imbalance by developing a library of scheduling techniques in two (GNU* and LLVM**) very popular and impacting OpenMP implementations

* https://bitbucket.org/PatrickABuder/libgomp/src/master/
** https://bitbucket.org/jhmkorndorfer/akan-master/src/master/libomp/
Take Away Messages

✧ Showed the effectiveness of state-of-the-art loop self-scheduling to address load imbalance due to application and system
✧ Existing OpenMP schedules (static, guided, dynamic) are good for corner cases

✧ More schedules are needed to add flexibility in the OpenMP runtime(s) against load imbalance by developing a library of scheduling techniques in two (GNU* and LLVM**) very popular and impacting OpenMP implementations
✧ FAC2 a good enough default scheduling in the observed cases; one can try out other schedules as well

* https://bitbucket.org/PatrickABuder/libgomp/src/master/
** https://bitbucket.org/jhmkorndorfer/akan-master/src/master/libomp/
Take Away Messages

✧ Showed the effectiveness of state-of-the-art loop self-scheduling to address load imbalance due to application and system
✧ Existing OpenMP schedules (static, guided, dynamic) are good for corner cases

✧ More schedules are needed to add flexibility in the OpenMP runtime(s) against load imbalance by developing a library of scheduling techniques in two (GNU* and LLVM**) very popular and impacting OpenMP implementations

✧ FAC2 a good enough default scheduling in the observed cases; one can try out other schedules as well

✧ How to improve performance beyond general scheduling techniques considering the limitations imposed by the OpenMP runtime system?
✧ Incorporate more runtime information to account for systemic-induced variability***
✧ Allow the user to develop specialized scheduling techniques regarding the needs of the application-system pair

* https://bitbucket.org/PatrickABuder/libgomp/src/master/
** https://bitbucket.org/jhmkorndorfer/akan-master/src/master/libomp/
***MOGSLib: Meta-programming-Oriented Global Scheduling Library: https://github.com/ECLScheduling/MOGLib
Ongoing Work

✧ **Toward A Standard Interface for User-Defined Scheduling in OpenMP**, iWOMP | Auckland | 2019
  ✧ A proposal for a standardized **User-Defined Scheduling (UDS)** interface
  ✧ Goal: allow the user to define custom scheduling techniques that can be specialized to the application needs.

✧ **Two-level Dynamic Load Balancing for High Performance Scientific Applications**, SIAM PP 2020
  ✧ Combination of dynamic loop self-scheduling in OpenMP and MPI to address the load imbalance at node-level and across nodes
  ✧ Goal: Leave no rank/thread idle while there is work
  ✧ Shown impressive benefits for astrophysics simulations

✧ Implement an *intelligent selection mechanism* among the many available options, based on previous work [Boulmier et al. 2017***; Banicescu et al. 2013****]

✧ *Upstream* the implemented self-scheduling methods to LLVM
Take Away Messages

✧ Showed the effectiveness of state-of-the-art loop self-scheduling to address load imbalance due to application and system
✧ Existing OpenMP schedules (static, guided, dynamic) are good for corner cases

✧ More schedules are needed to add flexibility in the OpenMP runtime(s) against load imbalance by developing a library of scheduling techniques in two (GNU* and LLVM**) very popular and impacting OpenMP implementations

✧ FAC2 a good enough default scheduling in the observed cases; one can try out other schedules as well

✧ How to improve performance beyond general scheduling techniques considering the limitations imposed by the OpenMP runtime system?
✧ Incorporate more runtime information to account for systemic-induced variability***
✧ Allow the user to develop specialized scheduling techniques regarding the needs of the application-system pair

* https://bitbucket.org/PatrickABuder/libgomp/src/master/
** https://bitbucket.org/jhmkorndorfer/akan-master/src/master/libomp/
***MOGSLib: Meta-programming-Oriented Global Scheduling Library: https://github.com/ECLScheduling/MOGLib
Dynamic Load Balancing of Loops in OpenMP

Florina M. Ciorba

SC19 Denver, 19.11.2019

11:15 am
OpenMP Booth #1737

University of Basel, Switzerland