LCPC 2021

Performance Evaluation of OSCAR Multi-target Automatic Parallelizing Compiler on Intel, AMD, Arm and RISC-V Multicores

Birk Martin Magnussen, Tohma Kawasumi, Hiroki Mikami, Keiji Kimura, Hironori Kasahara

Department of Computer Science and Engineering
Waseda University

Motivation

- Increasing number and types of cores in shared memory multicore architectures
  - Need for multi target-architecture parallelizing compiler
  - Need for little tuning required per-target configuration

- OSCAR automatic parallelizing compiler
  - Source-to-source compiler
    - Generated source can be compiled by both OpenMP and sequential compilers for different architectures
  - Multigrain parallelization; coarse-grain task parallelism, loop-level parallelism, statement-level parallelism

The OSCAR Compiler – Overview

To improve effective performance, cost-performance and software productivity and reduce power

Multigrain Parallelization (LCPC1991,2001,04)

coarse-grain parallelism among loops and subroutines (2000 on SMP), near fine grain parallelism among statements (1992) in addition to loop parallelism

Data Localization

Automatic data management for distributed shared memory, cache and local memory

(Local Memory 1995, 2016 on RP2,Cache2001,03)

Software Coherent Control (2017)

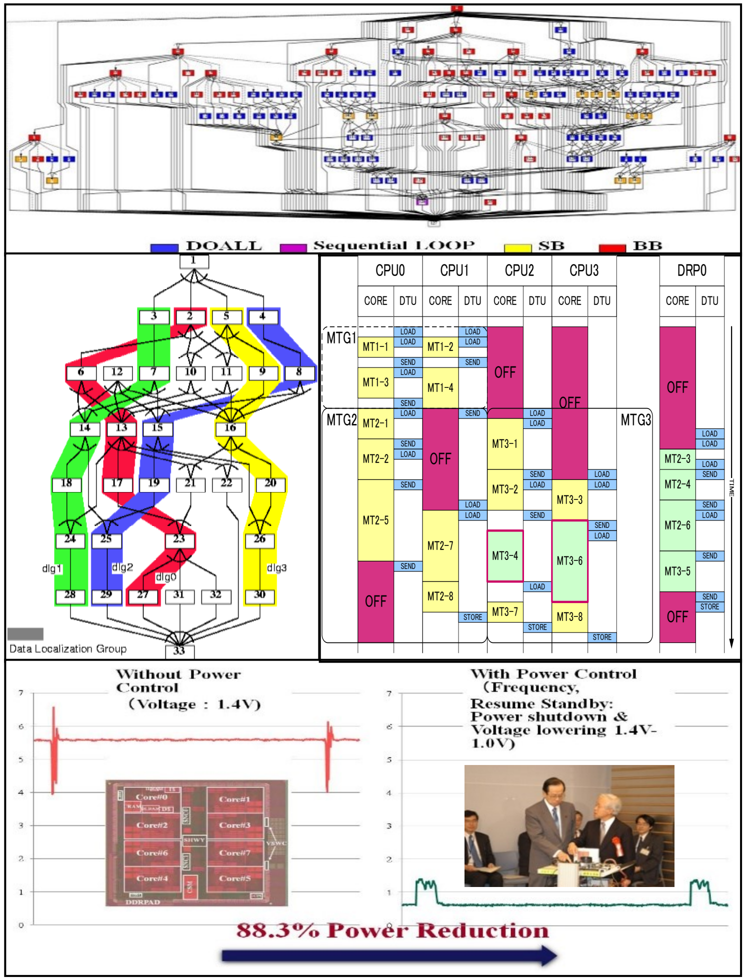
Data Transfer Overlapping (2016 partially)

Data transfer overlapping using Data Transfer Controllers (DMAs)

Power Reduction

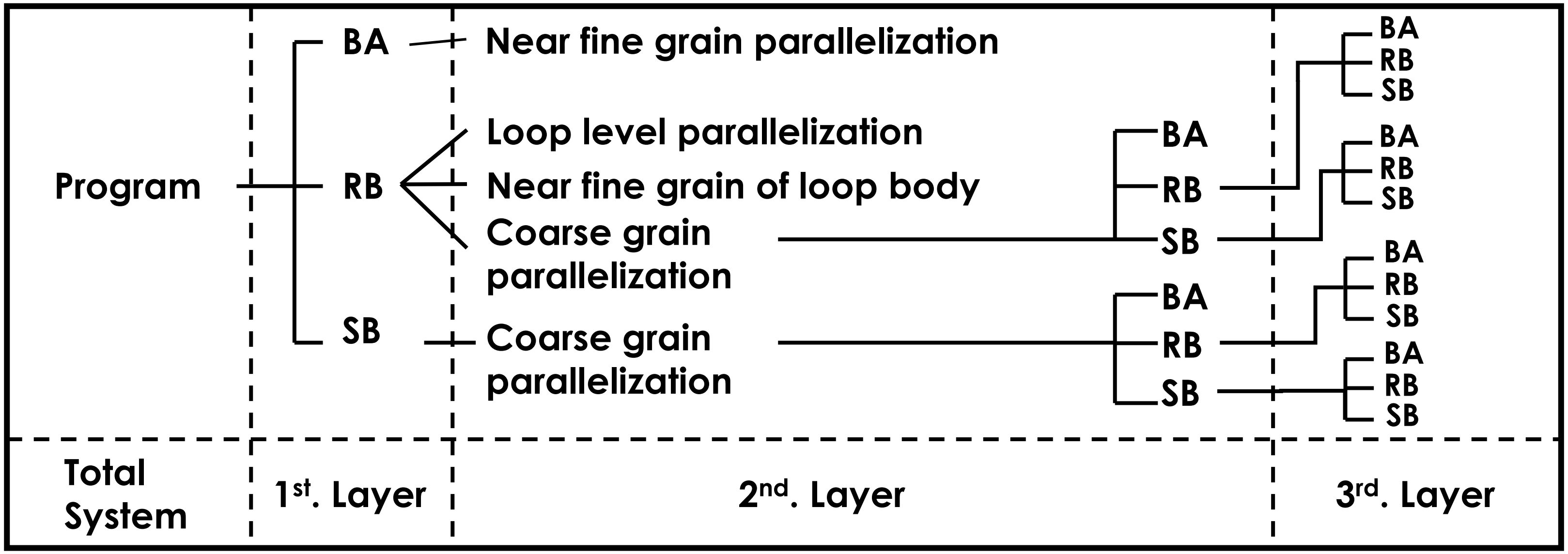
(2005 for Multicore, 2011 Multi-processes, 2013 on ARM)

Reduction of consumed power by compiler control DVFS and Power gating with hardware supports.

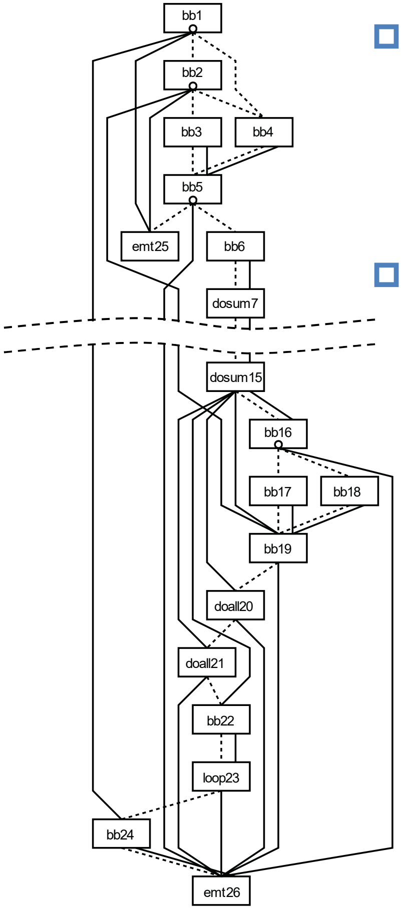


The OSCAR Compiler – Multigrain parallelism

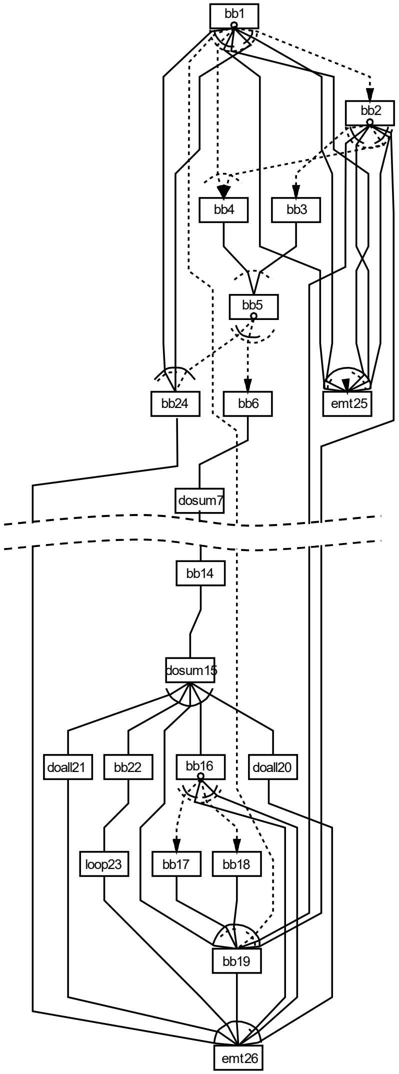
- macro-tasks in multigrain parallelism
  - BA: Block of Assignments
  - RB: Repetition Block
  - SB: Subroutine Block


Program
BA
RB
SB
Near fine grain parallelization
Loop level parallelization
Near fine grain of loop body
Coarse grain parallelization
Coarse grain parallelization
BA
RB
SB
BA
RB
SB
BA
RB
SB
BA
RB
SB
BA
RB
SB
BA
RB
SB
Total System
1st. Layer
2nd. Layer
3rd. Layer

The OSCAR Compiler – Macro-task generation



- Macro-Flow Graph (left)
  - —— : data dependency
  - - - - - : control flow
  - ○ : branch
- Macro-Task Graph (right)
  - —— : data dependency
  - - - - - : control dependency
  - ○ : branch
  - → : original control flow
  -) : AND
  - ⁞ : OR



- Earliest-execution conditions
  - Completion of directly data dependent tasks
  - Completion of tasks so that control flow passes task guaranteed

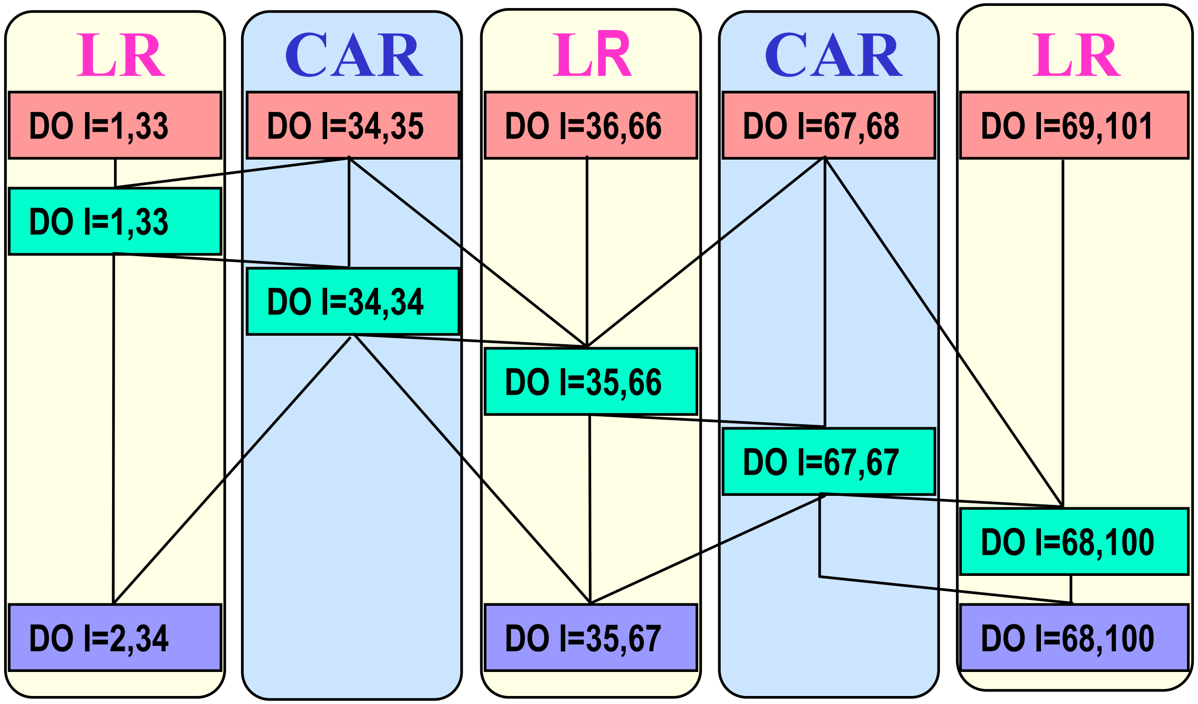
The OSCAR Compiler – Loop aligned decomposition

- loop-aligned decomposition

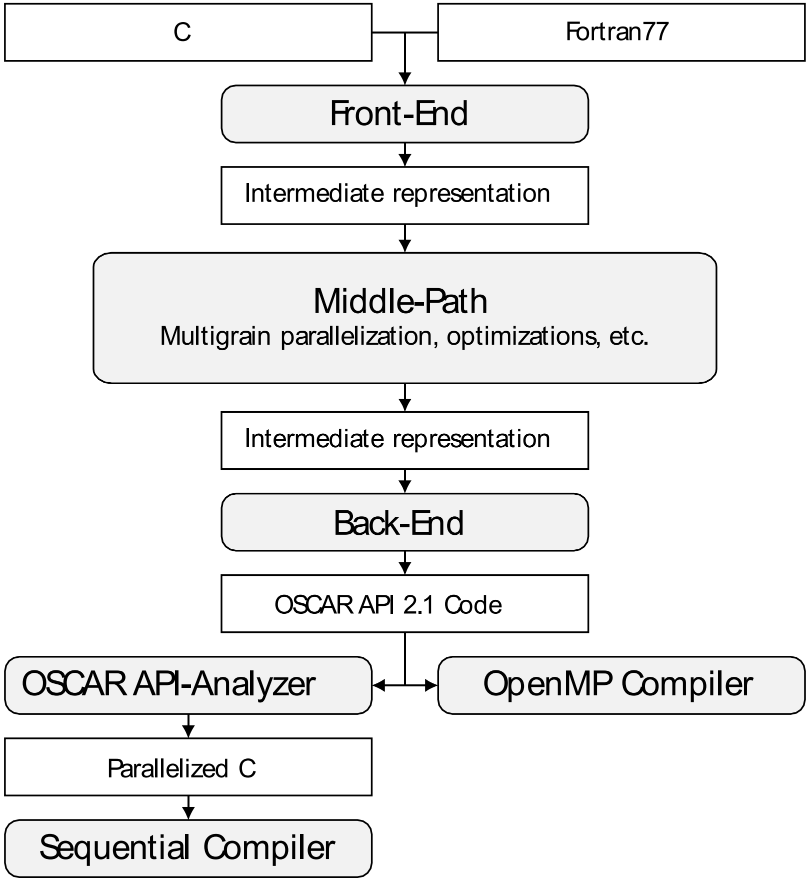
```
C RB1(Doall)
  DO I=1,101
   A(I)=2*I
  ENDDO
```

```
C RB2(Doseq)
  DO I=1,100
   B(I)=B(I-1)
     +A(I)+A(I+1)
  ENDDO
```

```
C RB3(Doall)
  DO I=2,100
   C(I)=B(I)+B(I-1)
  ENDDO
```



The OSCAR Compiler – Compile flow



C

Fortran77

Front-End

Intermediate representation

Middle-Path
Multigrain parallelization, optimizations, etc.

Intermediate representation

Back-End

OSCAR API 2.1 Code

OSCAR API-Analyzer

OpenMP Compiler

Parallelized C

Sequential Compiler

Investigated Benchmarks

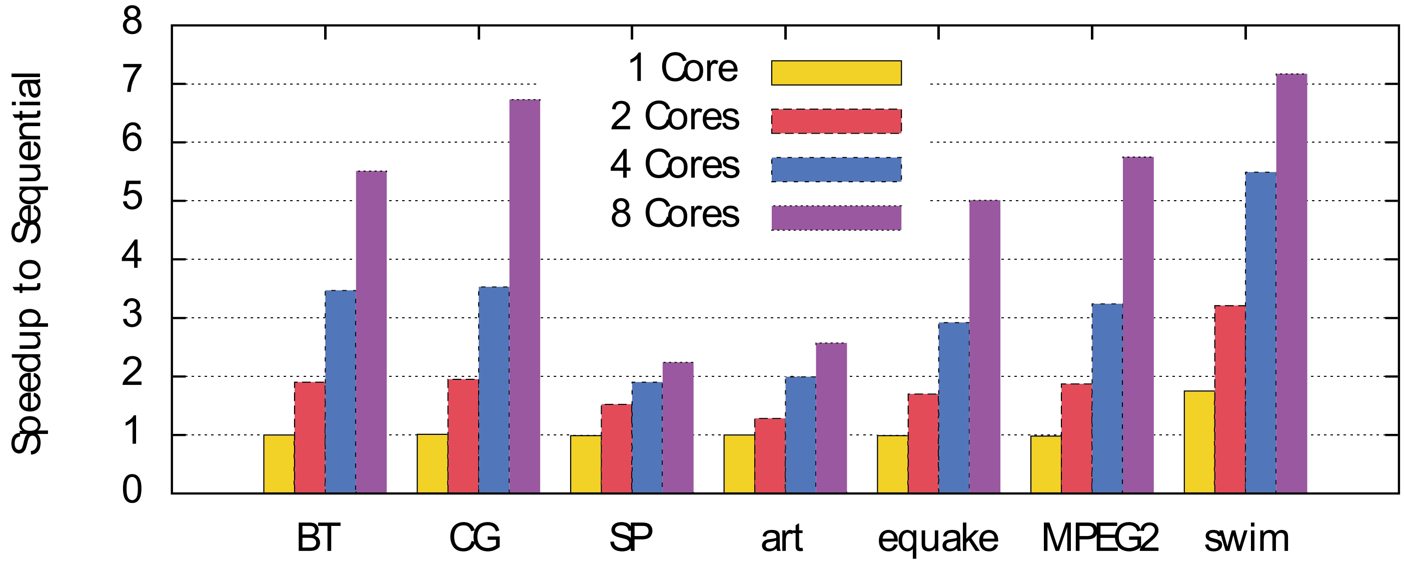
- NAS parallel benchmark suite
  - BT: Block Tri-diagonal solver
  - CG: Conjugate Gradient computation
  - SP: Scalar Penta-diagonal solver
- SPEC2000
  - art: Image recognition / Neural networks
  - equake: Seismic wave propagation simulation
  - swim: Shallow water modelling - Fortran 77
- MediaBench II
  - MPEG2 encoding

Intel Xeon E5-2650v4 – Statistics

- x86-64 based Architecture
- 12 Cores
- 2.2 GHz – 2.9 GHz
- 30 MiB shared L3 cache
  - L3 Cache: Shared by all cores

Intel Xeon E5-2650v4 – Benchmark results

- speedup to sequential version (higher is better)
- gcc as backend



- swim shows superlinear speedup and 1 core speedup
  - seq.: 58.1 s, 1 core OSCAR: 33.2 s, 4 core OSCAR: 10.5 s

Intel Xeon E5-2650v4 – swim cache statistics

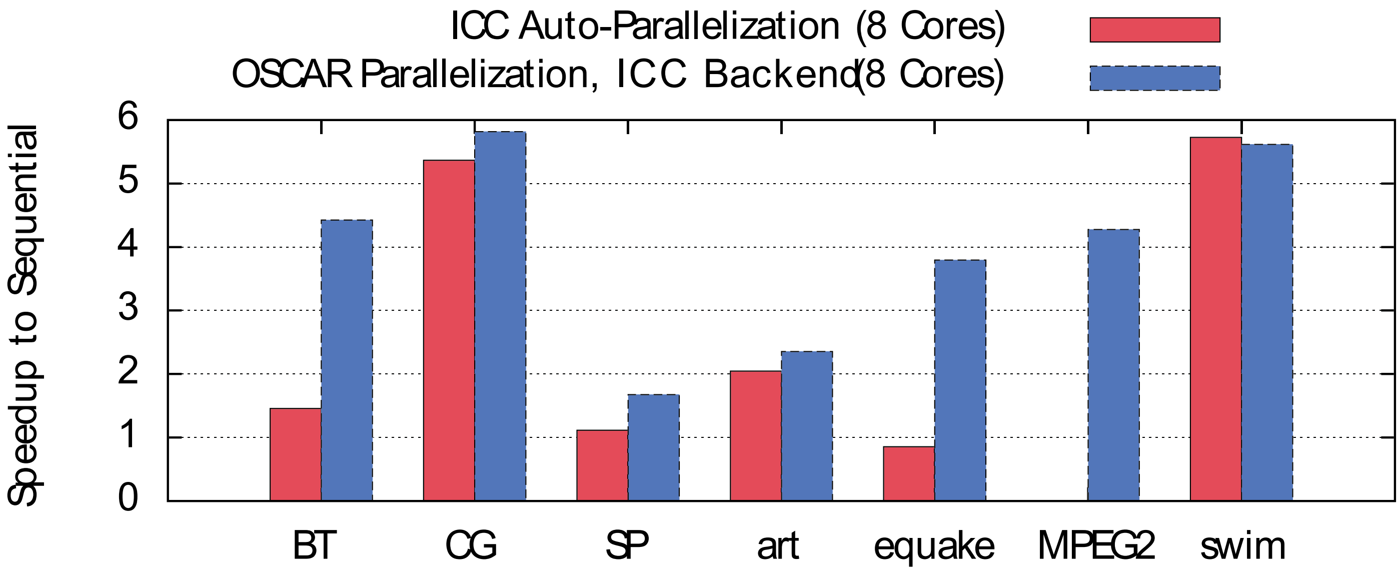
- swim benchmark cache statistics

Program	L1 loads	L1 load misses	L3 loads	L3 load misses
Sequential	$2.3 \cdot 10^{11}$	$1.2 \cdot 10^{11}$	$5.7 \cdot 10^{10}$	$1.1 \cdot 10^{10}$
OSCAR 1 core	$2.3 \cdot 10^{11}$	$6.5 \cdot 10^{10}$	$1.5 \cdot 10^{10}$	$8.2 \cdot 10^{9}$
OSCAR 2 cores	$2.2 \cdot 10^{11}$	$6.5 \cdot 10^{10}$	$1.5 \cdot 10^{10}$	$7.1 \cdot 10^{9}$
OSCAR 4 cores	$2.2 \cdot 10^{11}$	$6.5 \cdot 10^{10}$	$1.4 \cdot 10^{10}$	$6.1 \cdot 10^{9}$
OSCAR 8 cores	$2.2 \cdot 10^{11}$	$6.5 \cdot 10^{10}$	$1.3 \cdot 10^{10}$	$4.1 \cdot 10^{9}$

- reduction in L3 loads and misses
  - cause for speedup in comparison to sequential code

Intel Xeon E5-2650v4 – ICC backend compiler

- speedup to sequential version (higher is better)



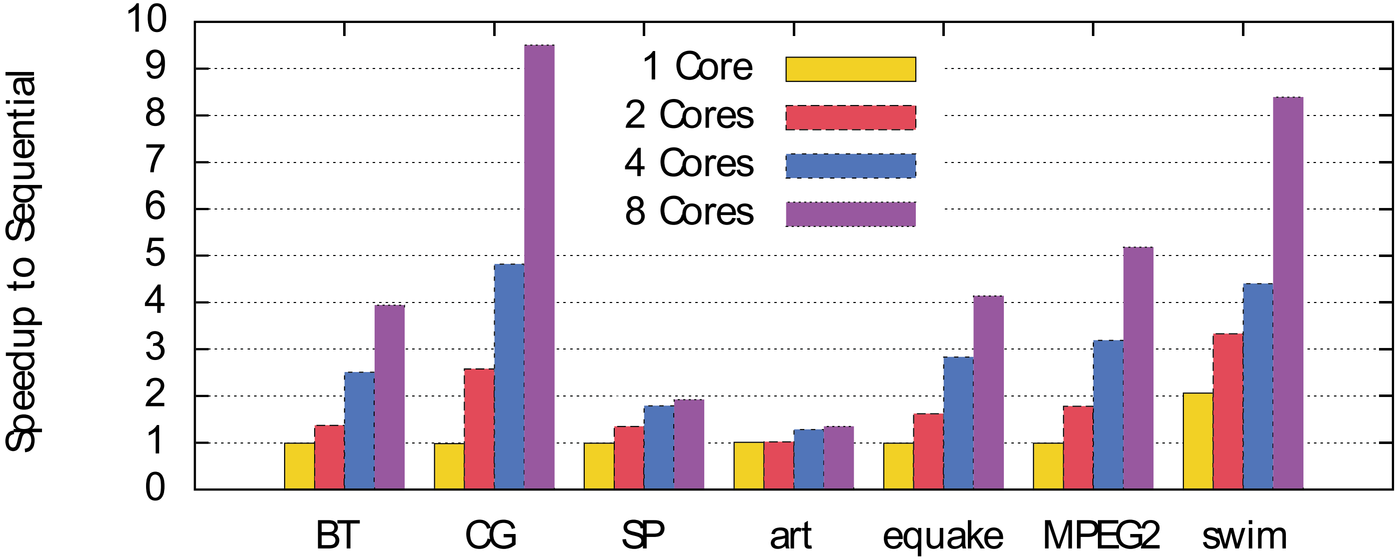
- OSCAR can be used with different backend compilers
  - slightly lower relative speedups are due to better sequential performance

AMD EPYC 7702P – Statistics

- x86-64 based Architecture
- 64 Cores
- 2.0 GHz – 3.35 GHz
- 16 MiB L3 cache per 4 core cluster
  - shared within the cluster

AMD EPYC 7702P – Benchmark results

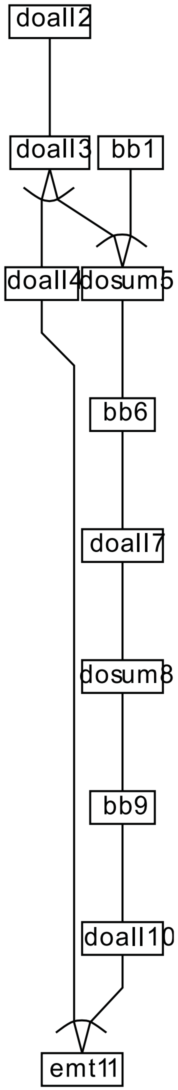
- speedup to sequential version (higher is better)
- gcc as backend



- CG and swim show superlinear speedup
  - CG: seq.: 0.86 s, 8 core OSCAR: 0.09 s

AMD EPYC 7702P – CG macrotask graph

- CG shows superlinear speedup
- macro-task graph consists of sequence of parallelized loops
  - good target for loop-aligned decomposition

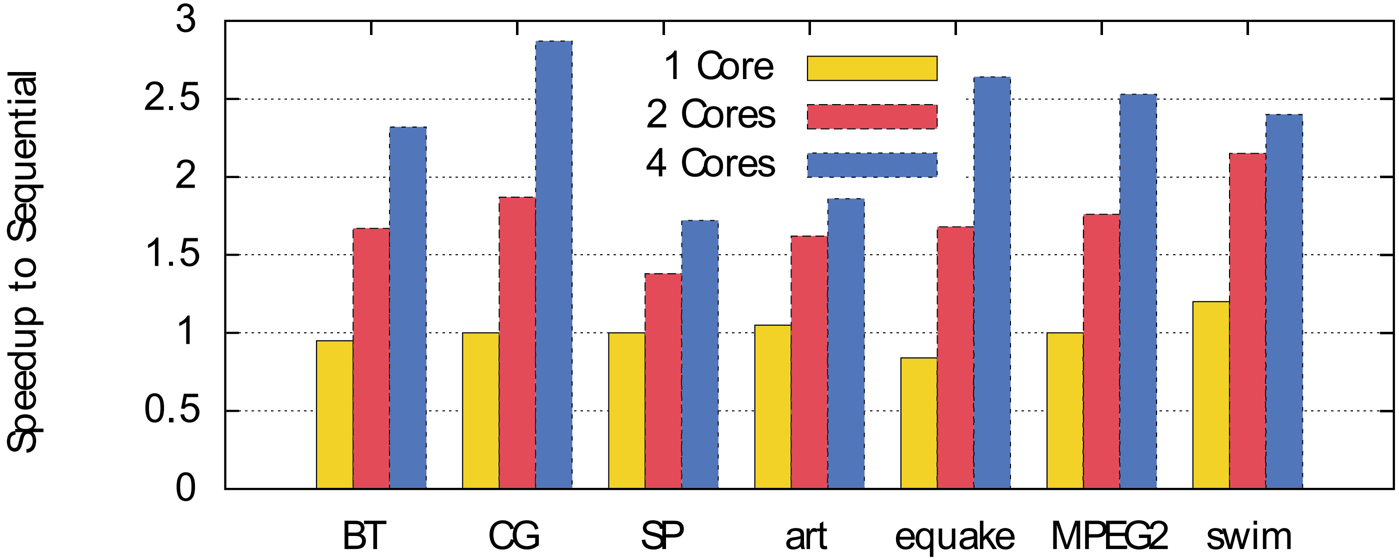


NVIDIA Carmel ARMv8.2 – Statistics

- Arm v8.2 based Architecture
- 6 Cores
- 1.4 GHz
- 4 MiB shared L3 cache
    - L3 Cache: Shared across all cores

NVIDIA Carmel ARMv8.2 – Benchmark results

- speedup to sequential version (higher is better)
- gcc as backend



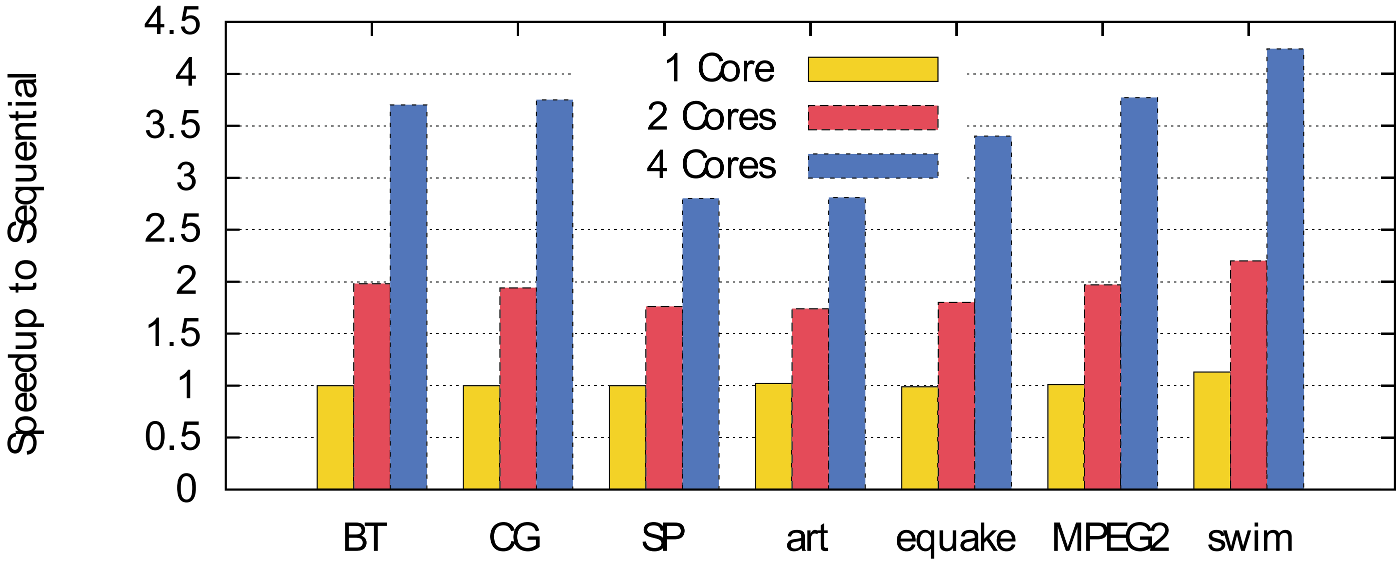
- overall good speedup is observed
  - equake: seq.: 19.0 s, 4 core OSCAR: 7.18 s

SiFive Freedom U740 – Statistics

- RISC-V based Architecture
- 4 Cores
- 1.2 GHz
- 2 MiB shared L2 cache
  - L2 Cache: Shared across all cores

SiFive Freedom U740 – Benchmark results

- speedup to sequential version (higher is better)
- gcc as backend



- overall good speedup is observed, swim superlinear
  - BT: seq.: 2041 s, 4 core OSCAR: 551 s

Conclusion

- The generated code shows high-performance on different modern architectures
  - Intel Xeon E5-2650v4 – swim – 8 cores: speedup 7.16
  - AMD EPYC 7702P – CG – 8 cores: speedup 9.5
  - NVIDIA Carmel – equake – 4 cores: speedup 2.64
  - SiFive Freedom U740 – mpeg2 – 4 cores: speedup 3.77
- This data shows that the OSCAR compiler can achieve good performance on various processor cores such as Intel x86, AMD x86, Arm and RISC-V
- Superlinear speedup was observed due to last level cache optimization with the loop-aligned decomposition
- OSCAR compiler can achieve this performance without any extensive per-system tuning