Compiler Enhanced Scheduling for OpenMP for Heterogeneous Multiprocessors

Jyothi Krishna V S and Shankar Balachandran

Presented by: Jyothi Krishna V S

IIT Madras

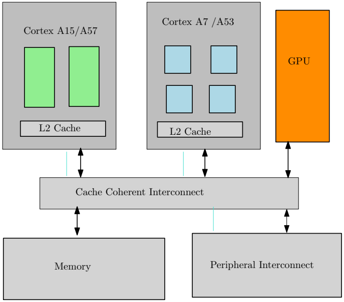
January 19, 2016

Overview

- `big.LITTLE`
- HMP Scheduling in `big.LITTLE`
- CES for `big.LITTLE`
- Mathematical Modelling
- OpenMP
    - `for` construct
    - `sections` construct
    - Thread Migration
- Results
- Conclusion

big.LITTLE

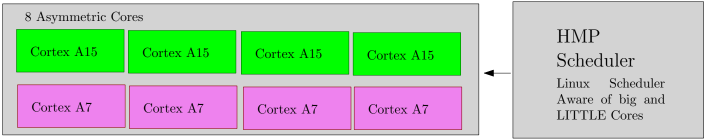
big.LITTLE System Design

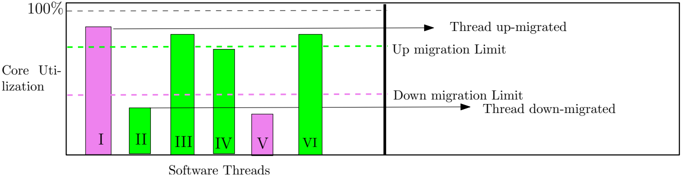


- Asymmetric Multicore Architecture from ARM
- Targets mobile platforms which has strict power constraints
- big-LITTLE migration ($30\mu S$) less expensive than DVFS state transition ($50 - 100\mu S$)

Core Types	Cortex-A7	Cortex-A15
Pipeline	simple 8-stage in-order	out-of-order, multi-issue
Frequency	600 - 1300 MHz	800 - 1900 MHz
Speed	1.9 DMIPS [?]	3.5-4.01 DMIPS
Instruction Set	Thumb-2	

big.LITTLE Scheduling: HMP Scheduling[1]





- Intergated in Linux kernel Complete Fair Scheduling (CFS)
- Scheduling based on history of utilization
- Threads with higher utilization scheduled in `big`

[1] http://www.arm.com/files/pdf/Heterogeneous_Multi_Processing_Solution_of_-Exynos_5_Octa_with_ARM_bigLITTLE_Technology.pdf

OpenMP API

- In C,C++ and FORTRAN.
- Team of threads executing parallel regions created by *master*.
- Barrier synchronized.
- Work sharing construct: unit of work executed by one of the threads.
  - omp_for: N iterations executed by the team. Iterations allocated based on static,dynamic or guided scheme.
  - omp_sections: set of sections, each executes once by one of the threads. Random allocation.
  - omp_single: One of the threads executes the contents inside the construct.

CES for big.LITTLE

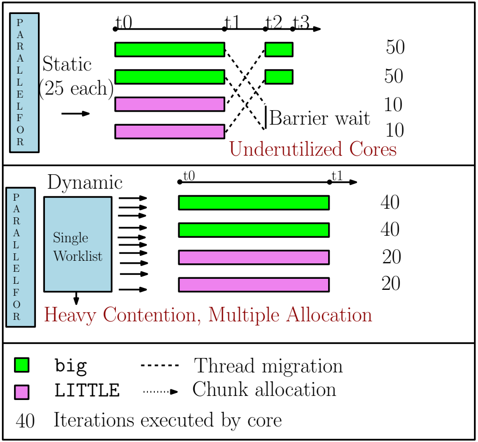
- OpenMP works well for SMP.
- Aim: Asymmetry aware compiler for reducing execution time and power consumption.
- Key Idea: Reduce the disparity in individual thread running time.
- Parallel-segment bounded by barriers.
- Optimize each parallel-segment separately.
- Running time of parallel-segment = running time of longest running thread.

Mathematical Modeling

- Total running cost of a thread : $T(ct) = T_{OP}(ct) + T_{MEM}(ct)$.
  - ct is core type.
- $T_{OP}(\texttt{big})/T_{OP}(\texttt{LITTLE})$ lower compared to $T_{MEM}(\texttt{big})/T_{MEM}(\texttt{LITTLE})$
- For multiple paths (branches): we take the largest cost.
- Value unknown at runtime : very high cost.
- The cost of a thread (once core is fixed) : $wl = select(T(\texttt{big}), T(\texttt{LITTLE}))$.
- Objective :$wl_{im} = min(max_{i,j \in team}|wl(i) - wl(j)|)$

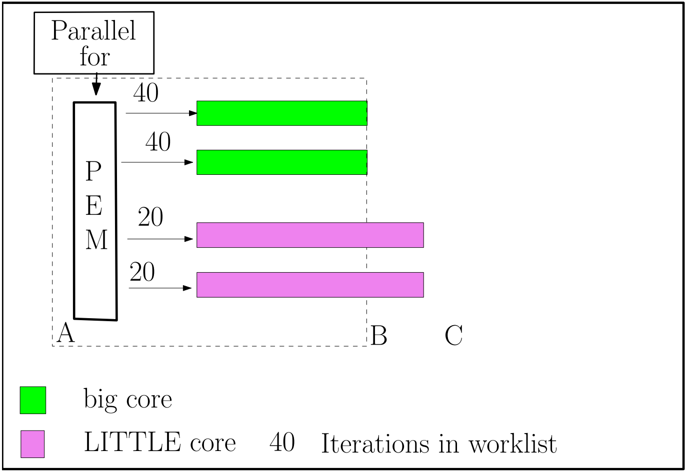
for Construct

- Current scheduling policy in HMP
  - Static : Thread migration and core under utilization
  - Dynamic : Multiple allocation and Heavy contention
  - Guided : Dynamic with varying chunk size



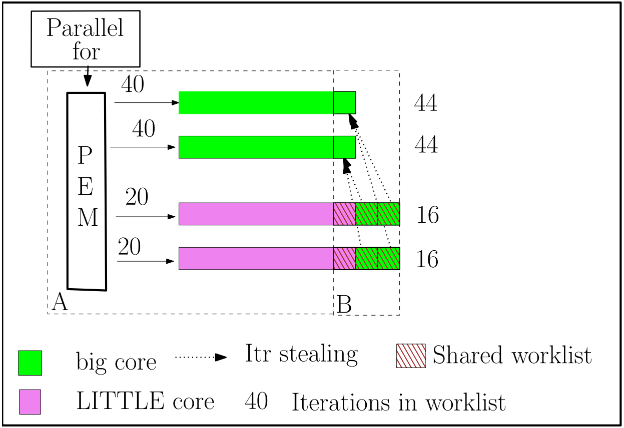
CES: Scheduling `for` Construct

- Initial thread to core scheduling fixed
- Each thread given a private worklist
- Initial iteration count based on the core type and iteration cost



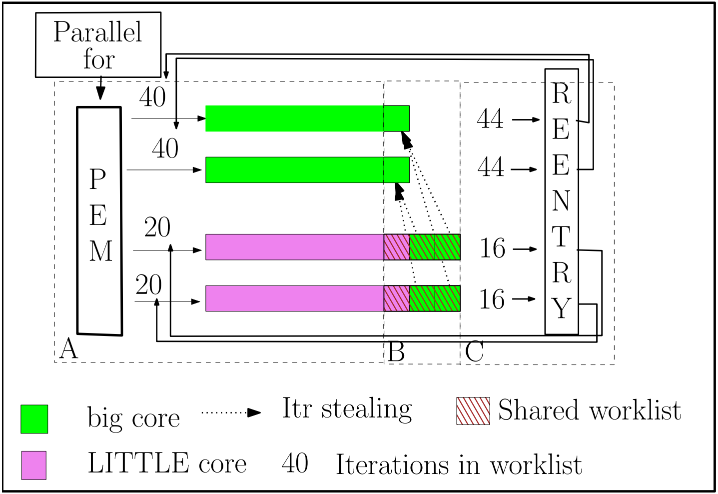
CES: `for` Construct - Stealing iterations

- Once finished start stealing form unfinished worklist
- Victim worklist is made shared
- Number of iterations stolen per chunk based on core types of stealer and victim



CES: `for` Construct for Nested Loops

- Special mechanism for `for` loops that are revisited
- Update initial division based on current execution

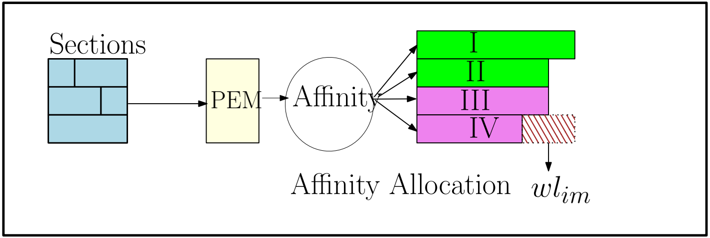


`sections` Construct

- Hetrogeneous workload
- OpenMP: Random allocation of sections to threads
- In CES, allocation is divided into two different stage
  - Affinity Allocation
  - Normalization Allocation

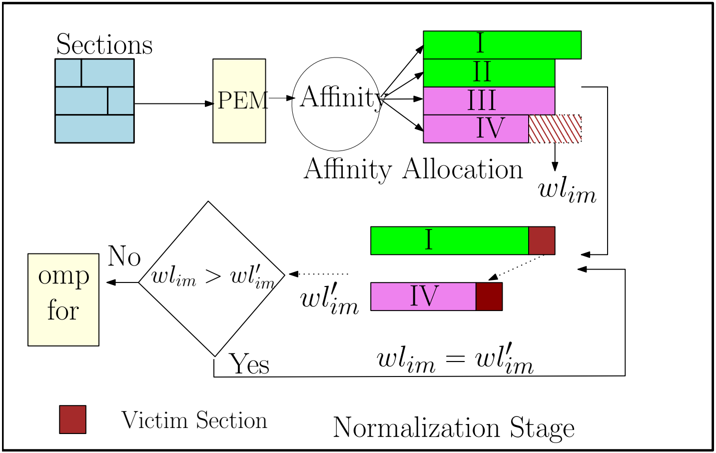
CES : sections Construct - Affinity allocation

- Thread to core mapping is fixed
- A section i, allocated to each core based on its affinity, aff_i
- $aff_i = T_i(big)/T_i(LITTLE)$
- Lower aff_i higher chance to be scheduled to big



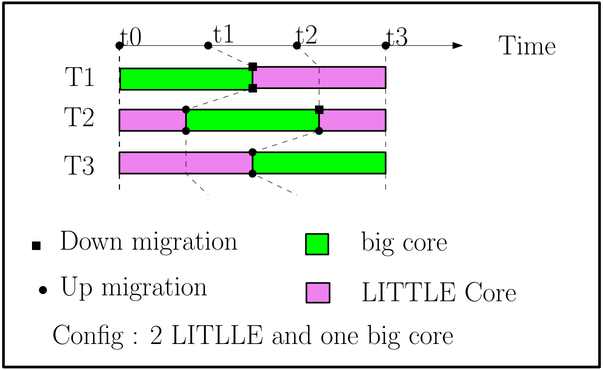
CES : `sections` Construct - Normalization allocation

- Normalization stage: reduce wl_{im} between threads with largest and lowest workloads



Thread Migration

- For equal workloads, where thread workload is fixed
- We induce thread migrations
- minimum guarantee point (mgp): Thread ready to be down migrated
- migration point: Time at which thread is up-migrated

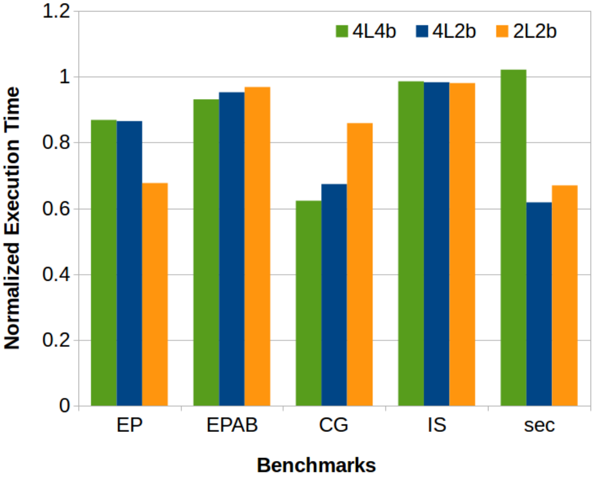


Implementation and Results

- Implemented using IMOP [2]
- Different configurations of big and LITTLE cores tried.
  - 4 big and 4 LITTLE cores.
  - 2 big and 4 LITTLE cores.
  - 2 big and 2 LITTLE cores.
- For different freqency configurations.
- NPB benchmarks
- EPAB: EP of NPB benchmark with re-visited for construct.
- sec: modified FSU sections benchmark added to demonstrate section scheduling.

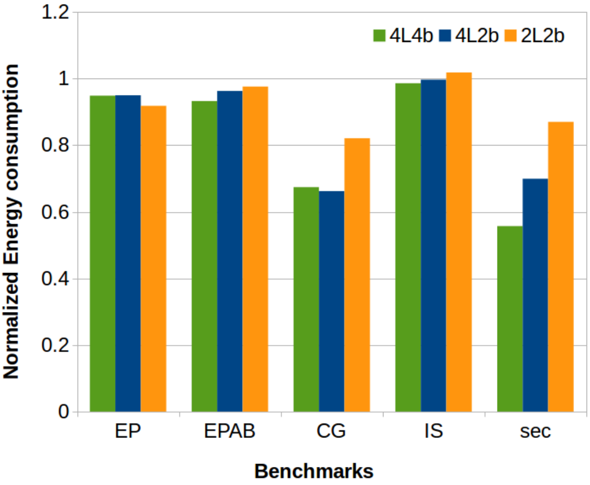
[2] Nougrahiya et al. IMOP : http://www.cse.iitm.ac.in/ amannoug/imop/

Execution time: big LITTLE config



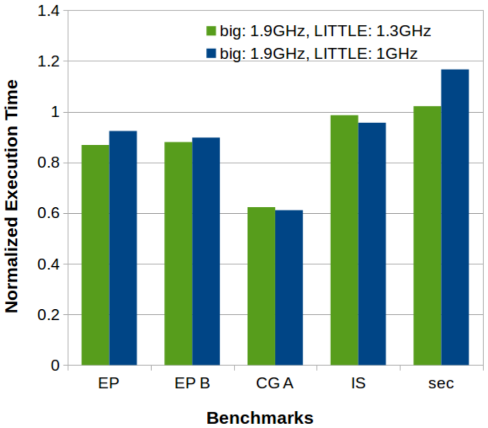
- sec shows higher execution time due to affinity scheduling.

Power



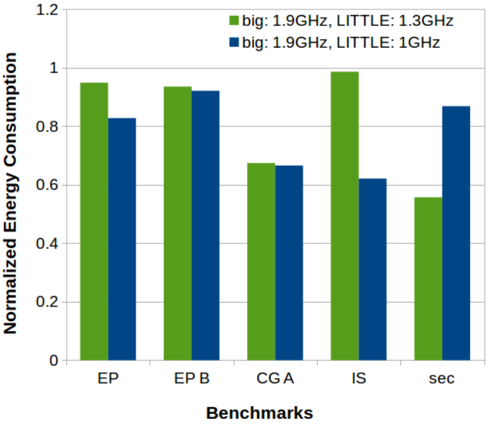
- Affinity scheduling helps sec with huge gain in energy consumption.

Varying frequencies of big and LITTLE



- f1 : big = 1.9 GHz LITTLE=1.3Ghz
- f2 : big = 1.9 Ghz LITTLE=1 Ghz

Varying frequencies of big and LITTLE



- IS initial division closer to optimal for f1.
- sec section allocation in f1 gives higher energy gains.

Conclusions

- Asymmetry aware compilation can produce a more efficient execution environment.
- CES optimizes for OpenMP for both homogeneous and heterogeneous workloads.
- On an average 18% improvement in execution time.
- 14% improvement in CPU power consumption.
- Future work involves handling more OpenMP constructs.
- Compilation for optimizing mobile applications.
- User handles for programmer to direct the optimizations.