UCIFF: Unified Clustering, Instruction scheduling and Fast Frequency selection for Heterogeneous Clustered VLIW

Vasileios Porpodas and Marcelo Cintra

University of Edinburgh

LCPC 2012

Scheduling, Scalability and Energy

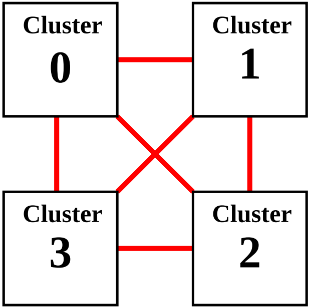
- Energy becomes a major design constraint
  - Dynamic Instruction scheduling in hardware consumes a large part of the energy budget
  - Statically scheduled processors are an energy-efficient alternative to dynamically scheduled processors
  - VLIW processors are high-performance statically scheduled processors

Scheduling, Scalability and Energy

- Energy becomes a major design constraint
  - Dynamic Instruction scheduling in hardware consumes a large part of the energy budget
  - Statically scheduled processors are an energy-efficient alternative to dynamically scheduled processors
  - VLIW processors are high-performance statically scheduled processors
- Resource scalability is necessary for both energy and performance
  - Clustered VLIW processors operate at an attractive energy-performance point
  - No global buses, only point-to-point communication
  - Instruction scheduling done by the compiler

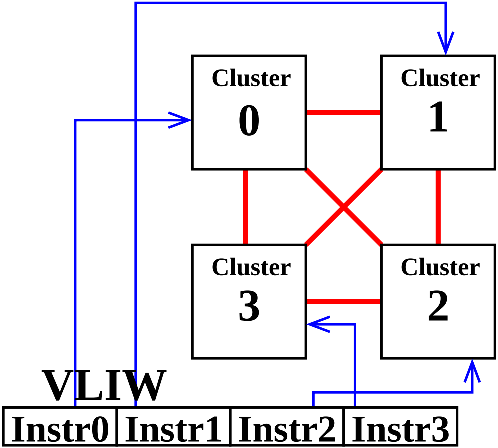
Clustered VLIW

- Statically scheduled
- Scalable
- Energy efficient
- Inter-Cluster delay



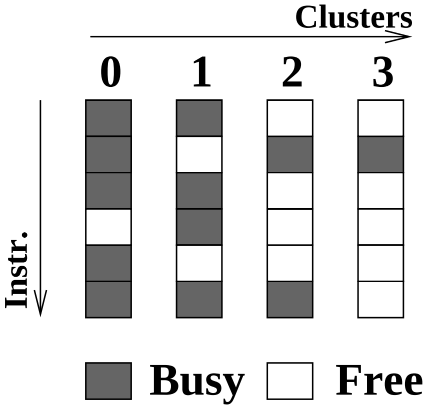
Clustered VLIW

- Statically scheduled
- Scalable
- Energy efficient
- Inter-Cluster delay
- Relies on compiler
- Explicit ILP



Cluster Utilization

- Few of the clusters are fully utilized

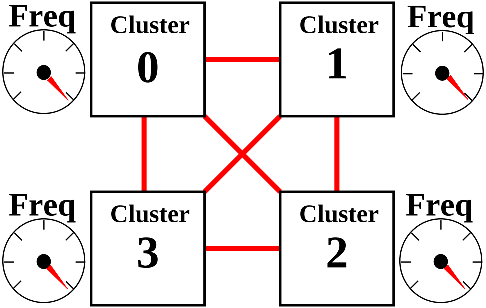


Cluster Utilization

- Few of the clusters are fully utilized
- Slack in schedule
- Opportunity to save energy without performance impact

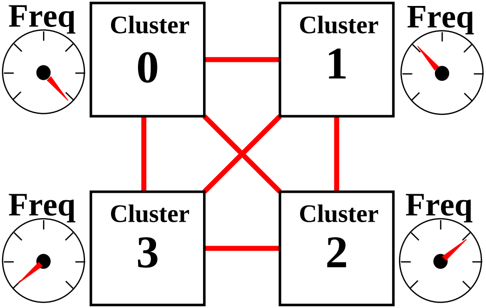
Heterogeneous Clustered VLIW

- Each cluster operates at its own frequency



Heterogeneous Clustered VLIW

- Each cluster operates at its own frequency
- Exploit cluster under-utilization
- Save energy by slowing down under-utilized clusters



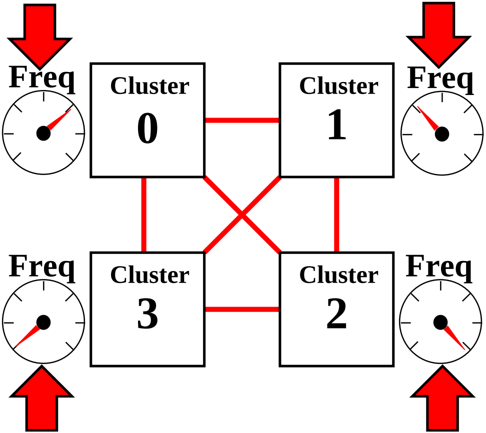
The Problem of Energy Efficiency on Clustered VLIW

- Cluster utilization varies
  - Some clusters are fully utilized
  - Others are running idle
- Per-cluster DVFS required for energy efficiency
  - Hardware DVFS not applicable (breaks semantics)
  - Effective compile-time DVFS required

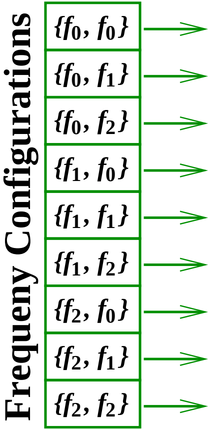
Outline

Problem Definition

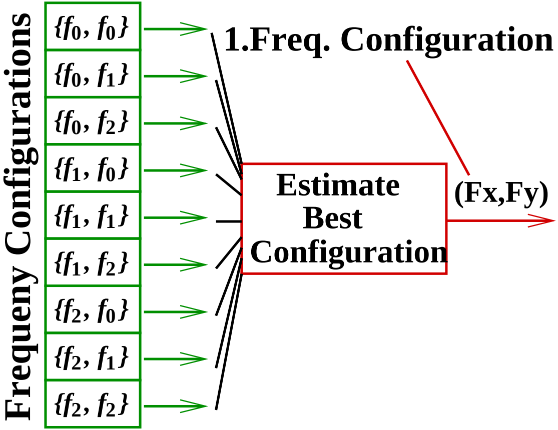
- How to determine "best" frequency for each cluster
- "Best" freq. is the one that leads to best Delay/Energy/ED/ED^2
- Determine frequencies during Instruction Scheduling



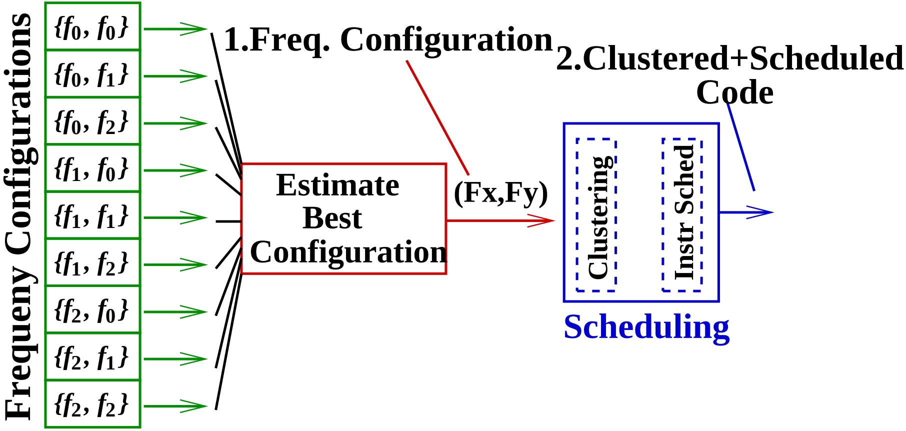
Existing Solution [CGO'07] (Decoupled)



Existing Solution [CGO'07] (Decoupled)

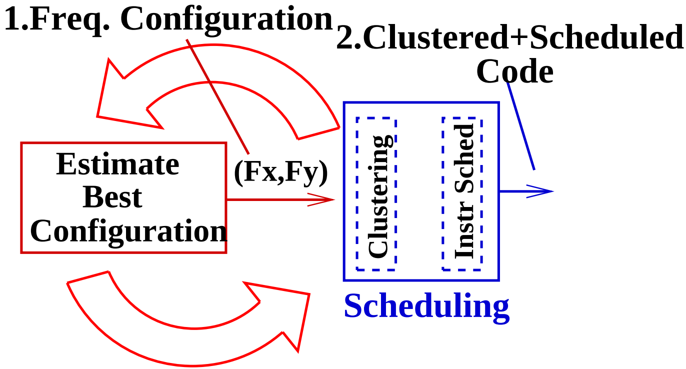


Existing Solution [CGO'07] (Decoupled)



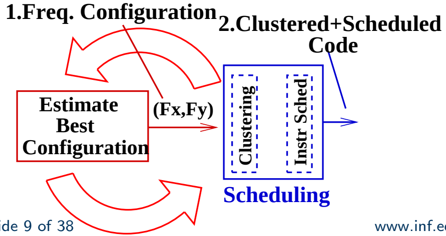
Existing Solution [CGO'07] (Decoupled)

- Phase ordering problem



Existing Solution [CGO'07] (Decoupled)

- Phase ordering problem
- Estimation of Best freq. requires knowledge of Performance and Energy
- Performance and Energy measurement requires schedule
- Scheduling requires that the frequencies are set



Outline

Introduction

Problem Definition and Existing Solutions

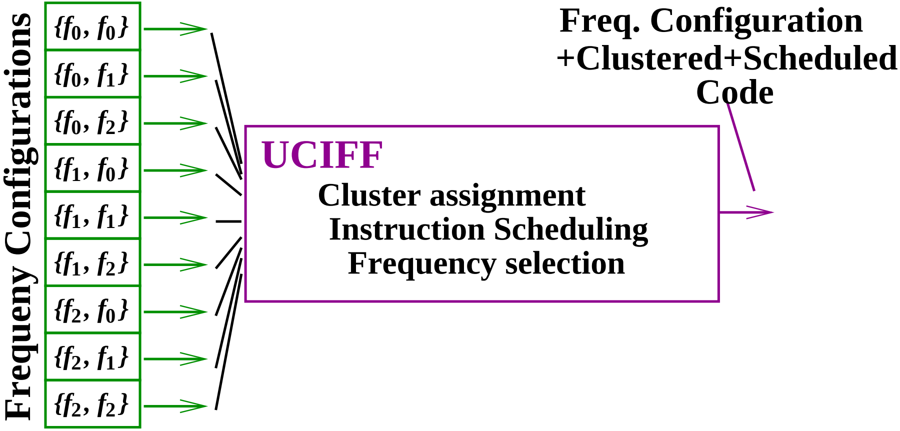
UCIFF

Experimental Setup and Results

Conclusion

UCIFF (unified)

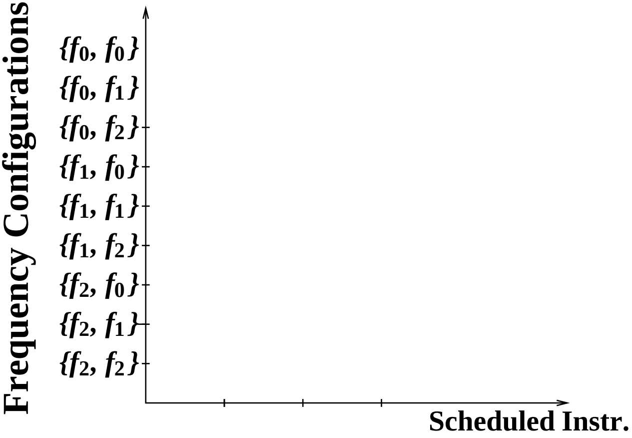
- No cyclic dependency



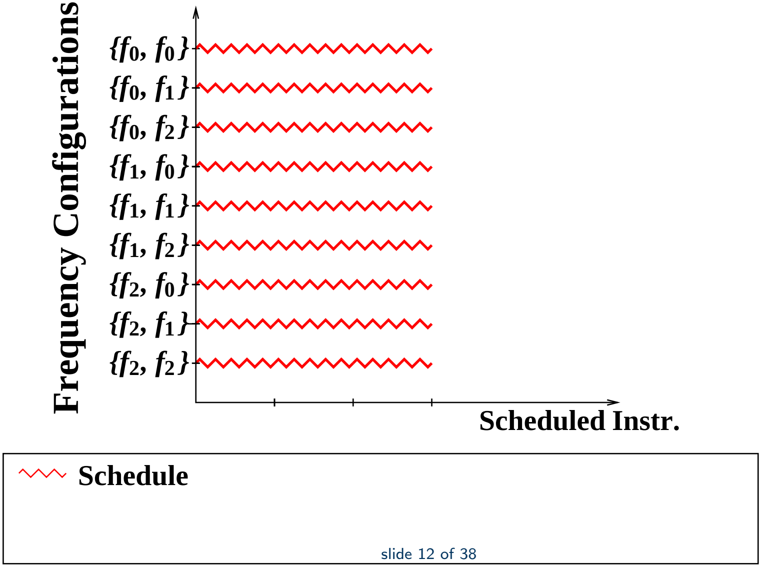
Full-Search Solution

- Brute-force: Try (schedule) all configurations
- After trying all find the best
- Obviously the slowest method

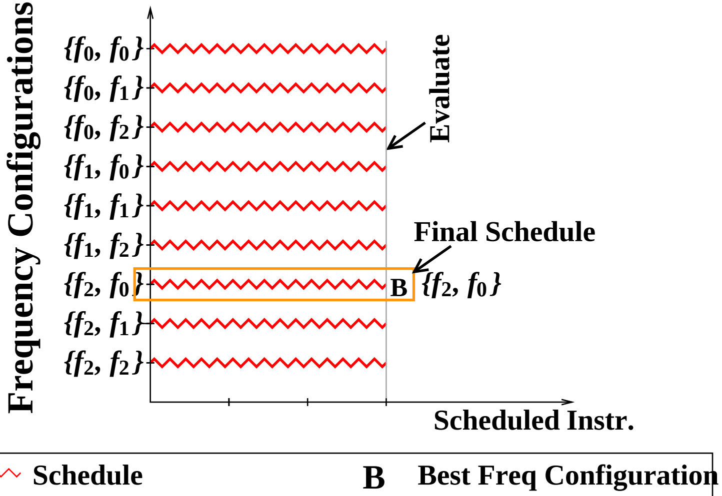
Full-Search Solution



Full-Search Solution



Full-Search Solution

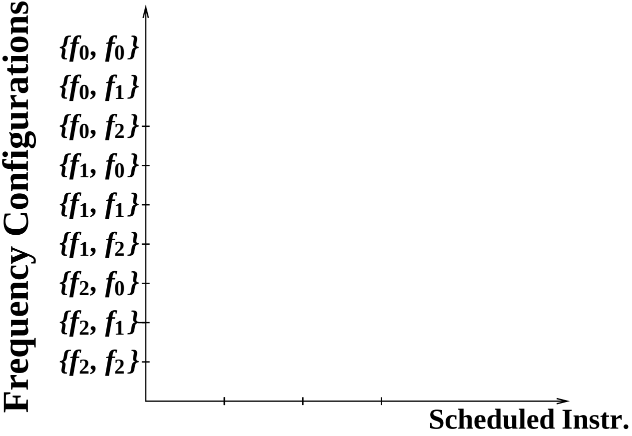


Theoretical Oracle Solution

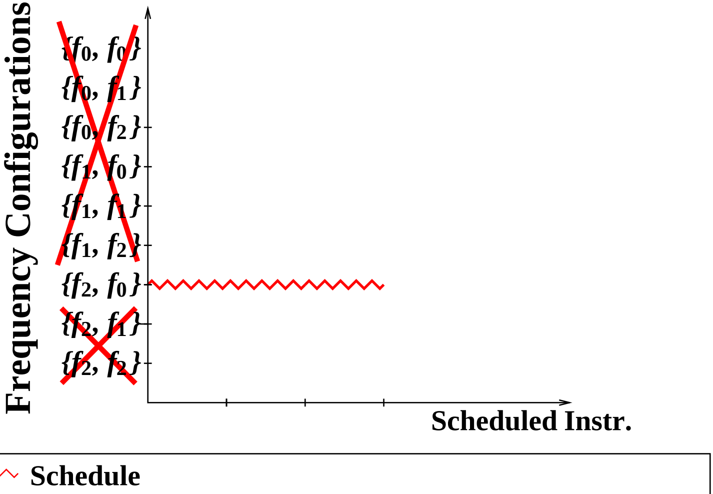
- A-priori knowledge of the best frequency configuration
- Schedules only the configuration which will generate the best schedule
- Fastest but Non-implementable

THE UNIVERSITY of EDINBURGH
informatics

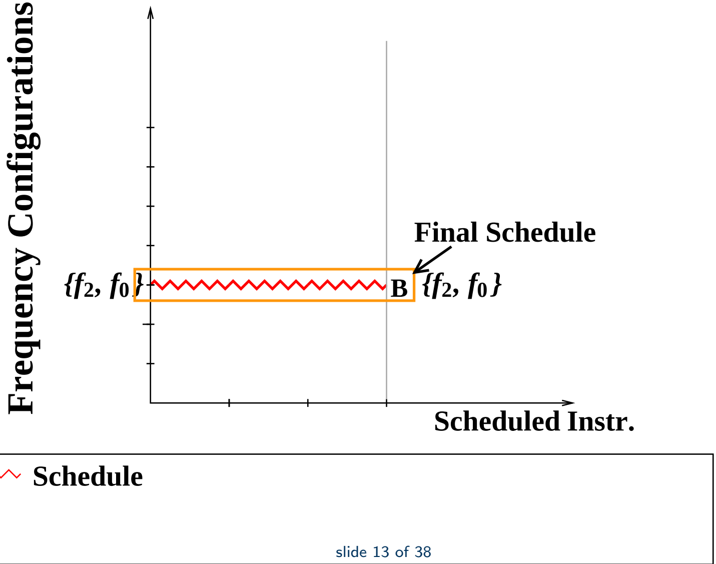
Theoretical Oracle Solution



Theoretical Oracle Solution



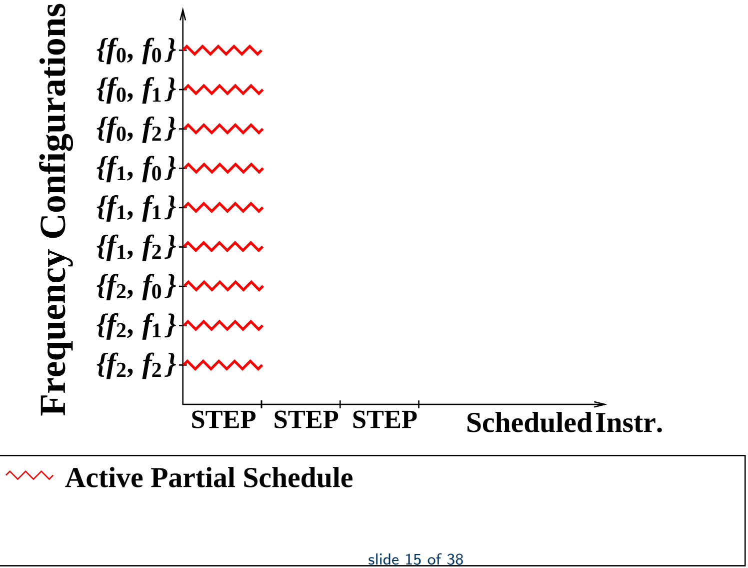
Theoretical Oracle Solution



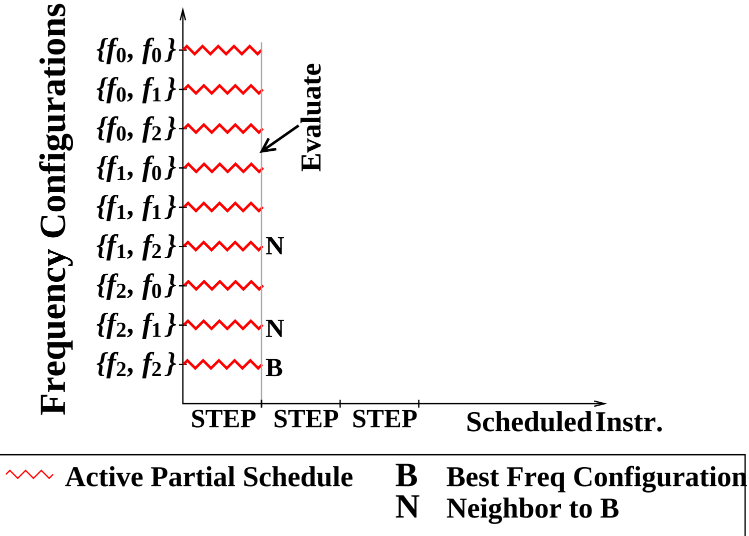
UCIFF

- Finds a good solution without resorting to full-search
  - Hill-Climbing over frequency configurations
  - Partial schedules “STEP” scheduling cycles each
  - At the end of each “STEP” it finds the best configuration and schedules it along with its neighbors for the next “STEP”

UCIFF


Frequency Configurations
{f0, f0}
{f0, f1}
{f0, f2}
{f1, f0}
{f1, f1}
{f1, f2}
{f2, f0}
{f2, f1}
{f2, f2}
STEP
STEP
STEP
Scheduled Instr.
Active Partial Schedule

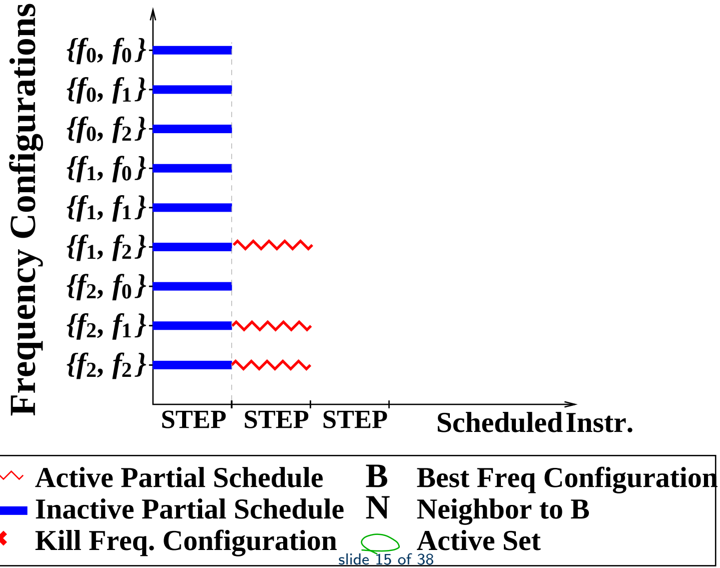
UCIFF


Frequency Configurations
{f0, f0}
{f0, f1}
{f0, f2}
{f1, f0}
{f1, f1}
{f1, f2}
{f2, f0}
{f2, f1}
{f2, f2}
Evaluate
N
N
B
STEP
STEP
STEP
Scheduled Instr.
Active Partial Schedule
B Best Freq Configuration
N Neighbor to B

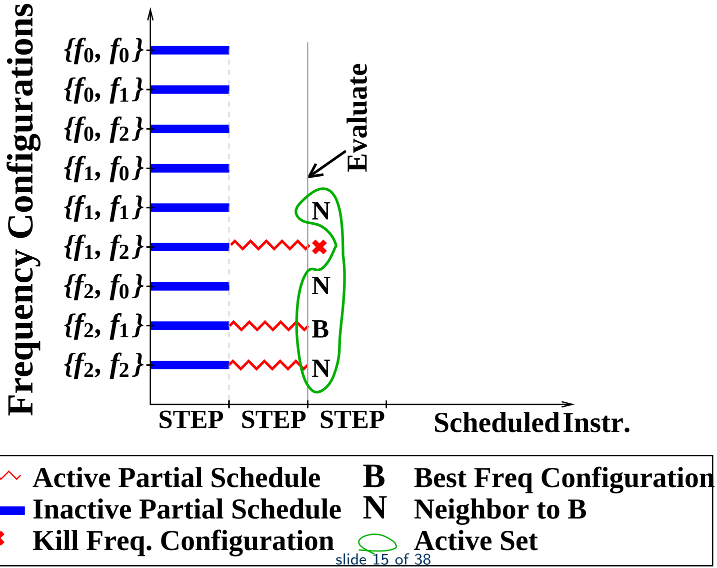
UCIFF


Frequency Configurations
{f0, f0}
{f0, f1}
{f0, f2}
{f1, f0}
{f1, f1}
{f1, f2}
{f2, f0}
{f2, f1}
{f2, f2}
N
N
B
STEP
STEP
STEP
Scheduled Instr.
Active Partial Schedule
Kill Freq. Configuration
B
Best Freq Configuration
N
Neighbor to B
Active Set

UCIFF

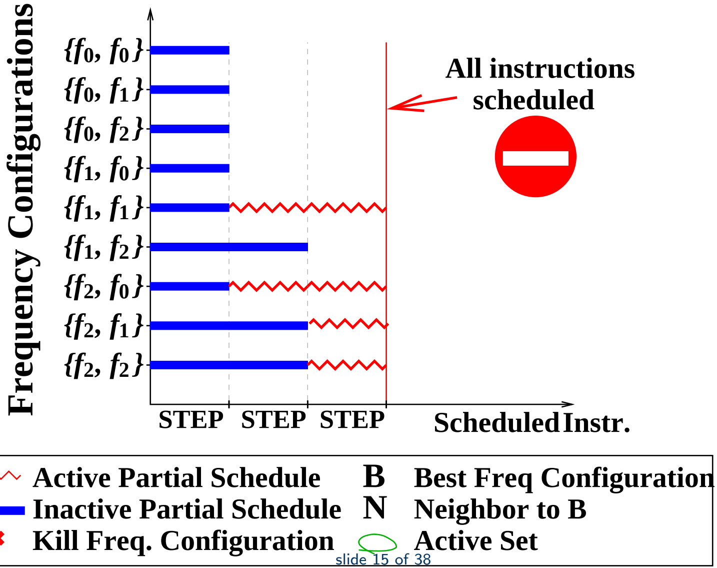

Frequency Configurations
{f0, f0}
{f0, f1}
{f0, f2}
{f1, f0}
{f1, f1}
{f1, f2}
{f2, f0}
{f2, f1}
{f2, f2}
STEP
STEP
STEP
Scheduled Instr.
Active Partial Schedule
Inactive Partial Schedule
Kill Freq. Configuration
B Best Freq Configuration
N Neighbor to B
Active Set

UCIFF

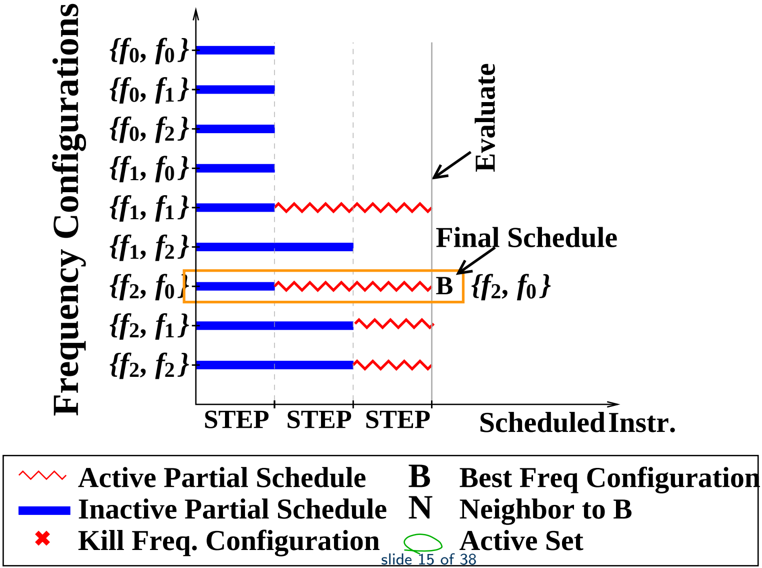


Frequency Configurations
{f0, f0}
{f0, f1}
{f0, f2}
{f1, f0}
{f1, f1}
{f1, f2}
{f2, f0}
{f2, f1}
{f2, f2}
Evaluate
N
N
B
N
STEP
STEP
STEP
Scheduled Instr.
Active Partial Schedule
Inactive Partial Schedule
Kill Freq. Configuration
B Best Freq Configuration
N Neighbor to B
Active Set

UCIFF


Frequency Configurations
{f0, f0}
{f0, f1}
{f0, f2}
{f1, f0}
{f1, f1}
{f1, f2}
{f2, f0}
{f2, f1}
{f2, f2}
All instructions scheduled
STEP
STEP
STEP
Scheduled Instr.
Active Partial Schedule
Inactive Partial Schedule
Kill Freq. Configuration
B
N
Best Freq Configuration
Neighbor to B
Active Set

UCIFF


Frequency Configurations
{f0, f0}
{f0, f1}
{f0, f2}
{f1, f0}
{f1, f1}
{f1, f2}
{f2, f0}
{f2, f1}
{f2, f2}
Evaluate
Final Schedule
B
{f2, f0}
STEP
STEP
STEP
Scheduled Instr.
Active Partial Schedule
Inactive Partial Schedule
Kill Freq. Configuration
B
Best Freq Configuration
N
Neighbor to B
Active Set

UCIFF

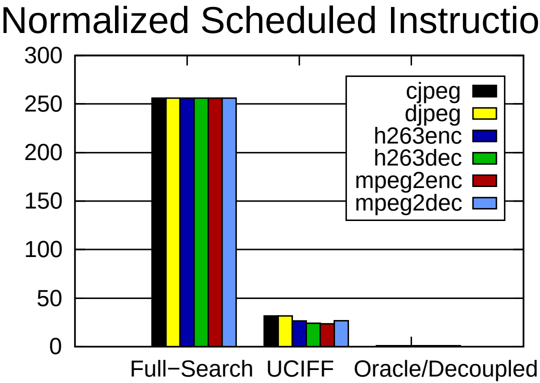
- Benefits of UCIFF hill-climbing:
  - More accurate frequency selection than CGO'07 (no phase-ordering problem)
  - Not based on estimation of performance or energy consumption, it measures the actual schedule.
  - Less scheduling time than full-search
  - Accuracy close to full-search

Outline

Experimental Setup

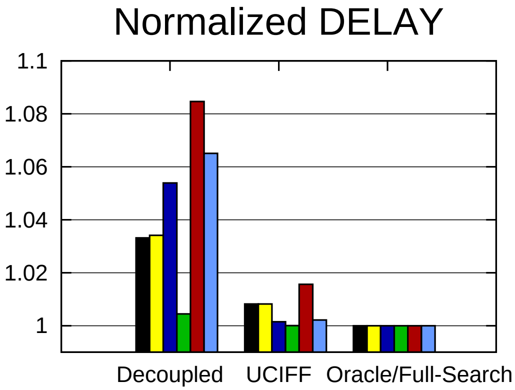
- Compiler
  - GCC-4.5.0
  - Modified Haifa-Scheduler
  - Energy model built into the scheduler
- Architecture
  - IA64-based 4-cluster/4-issue VLIW 1-cycle inter-cluster delay
  - 4 possible frequencies. Fastest:Slowest = 7:4
- Benchmarks
  - MediabenchII Video Benchmark suite
- Compare
  - Decoupled, Full-Search, Oracle, UCIFF

Time Complexity Comparison



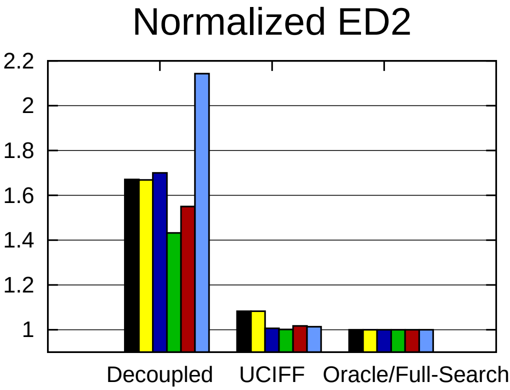
- 5× faster than Full-Search
- 30× slower than theoretical oracle

Delay Comparison



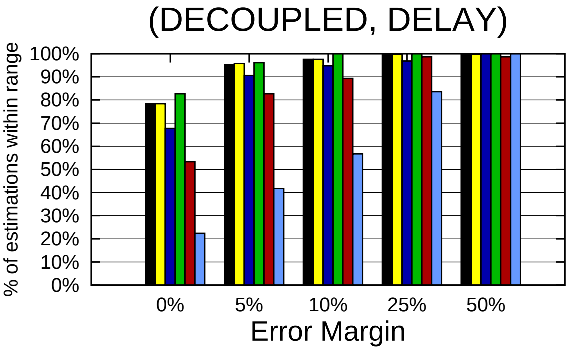
- Decoupled about 5% worse than Oracle
- UCIFF almost identical to Oracle
- Delay is biased towards high frequencies

ED^2 Comparison



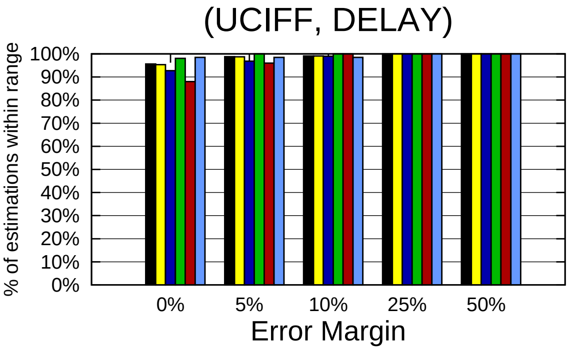
- Decoupled 1.6× worse than oracle
- UCIFF within 5% of Oracle
- ED^2 is hard to estimate

Delay Estimation Accuracy



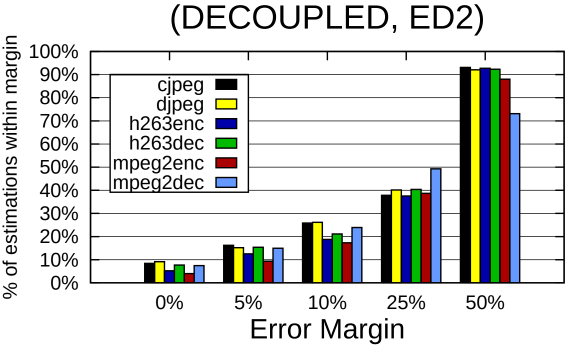
- Decoupled: only 60% of estimations are the same as Oracle

Delay Estimation Accuracy



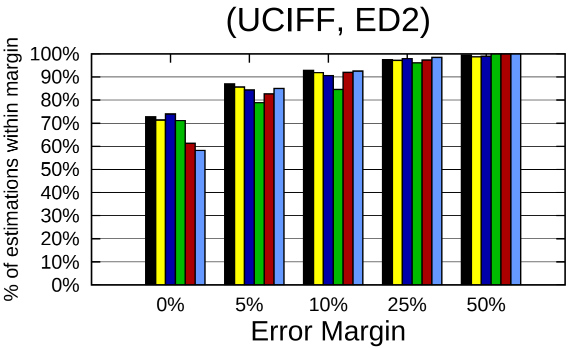
- Decoupled: only 60% of estimations are the same as Oracle
- UCIFF very close to Oracle (90%)

ED^2 Estimation Accuracy



- Decoupled is very inaccurate 40% of estimations within 25% of the Oracle

ED^2 Estimation Accuracy



- Decoupled is very inaccurate 40% of estimations within 25% of the Oracle
- UCIFF is at 95% within 25% of Oracle

Conclusion

- Proposed UCIFF, a unified scheduling algorithm that
  - Performs cluster assignment
  - Performs instruction scheduling
  - Selects cluster frequencies

  in a heterogeneous clustered VLIW
- UCIFF is more accurate and generates better schedules than the current state-of-the-art
- UCIFF is faster than Full-Search while generating code of equivalent quality

UCIFF: Unified Clustering, Instruction scheduling and Fast Frequency selection for Heterogeneous Clustered VLIW

Vasileios Porpodas and Marcelo Cintra

University of Edinburgh

LCPC 2012

Backup slides

- Bibliography
- CGO'07 Estimations
- Scheduling for heterogeneous (various freq.)
- UCIFF Energy Model
- DVFS regions
- UCIFF Neighbors
- UCIFF Algorithm

Bibliography

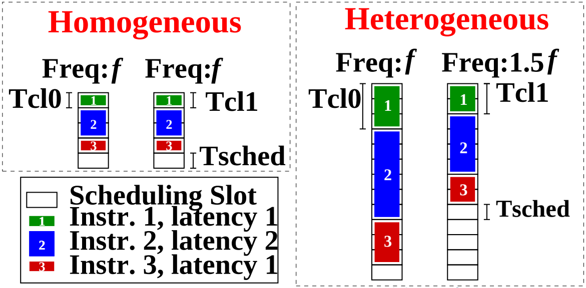
- [CGO'07] A. Aleta, J. Codina, A. Gonzalez, and D. Kaeli. Heterogeneous clustered vliw microarchitectures. In CGO, pages 354-366, 2007.

CGO'07 Energy & Performance Estimation

- Performance Estimation:
  1. Perform Scheduling on a homogeneous architecture
  2. Cycles = cycles of homogeneous multiplied by the arithmetic mean of the clock periods of the heterogeneous clusters: $Time = cycles_{hom} \times (\sum_{cl} T_{cl})/NumOfClusters$
- Energy Estimation (similar to UCIFF except:)
  1. Dynamic energy of cluster is equal to a fraction of that of the homogeneous cluster, proportional to the ratio of the cluster's frequency to the average frequency: $E_{dyn,ins}(cl) = E_{dyn,ins\_hom}(cl) \times f_{cl}/[\sum_{cl}(f_{cl})/NumOfClusters]$
  2. Energy of interconnect is equal to that of the homogeneous $E_{dyn,icc} = P_{icc} \times NumICCs_{homogeneous}$

Scheduling for clusters of various Freq.

- Scheduler's internal frequency is the lowest integer common multiple of all possible frequencies of all clusters (T_{sched})
- Instruction latencies are specific to each cluster and are a multiple of the original latencies: $T_{cl}/T_{sched} \times Original\_Latency$



DVFS Region

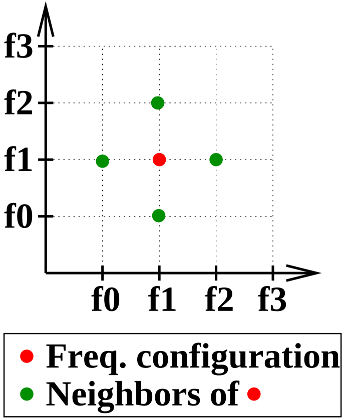
- Scheduling regions are too small for DVFS (H/W limitation)
- Possible solutions:
  - Micro-Architecture: Push DVFS points into a FIFO queue and take the average
  - Software1: Sampling at a rate acceptable by H/W
  - Software2: Get an average single DVFS point for the whole program

UCIFF Energy Model

- Total Energy:
  - $E = \sum_{clusters}[E_{st}(cl) + E_{dyn}(cl)]$
- Static Energy:
  - $E_{st}(cl) = P_{st} \times cycles_{cl} \times T_{cl}$
  - $P_{st}(cl) = C_{st} \times V_{cl}$
- Dynamic Energy:
  - $E_{dyn}(cl) = E_{dyn,ins}(cl) + E_{dyn,icc}$
  - $E_{dyn,ins}(cl) = \sum_{ins}[P_{ins}(cl) \times Latency(ins, cl)]$
  - $P_{ins}(cl) = C_{dyn} \times f_{cl} \times V_{cl}^2$
  - $E_{dyn,icc} = P_{icc} \times NumICCs$
  - $P_{icc} = C_{dyn} \times f_{fastest} \times V_{fastest}^2$

UCIFF Neighbors

- The Configuration $\{f_{na}, f_{nb}, f_{nc}, \ldots\}$ is a UCIFF neighbor of $\{f_a, f_b, f_c, \ldots\}$ if $nx = x$ for all x except one (say y) such that $|ny - y| < NDistance$.



```
/* Unified Cluster assignment Instr. Scheduling and Fast Frequency selection.
   In1: METRIC_TYPE that the scheduler should optimize for.
   In2: Schedule STEP instructions before evaluating and getting the best.
   In3: STEPVAR: Decrement STEP by STEPVAR upon each evaluation.
   In4: NEIGHBORS: The number of neighbors per cluster.
   Out: Scheduled Code and Best Frequency Configuration. */
uciff (METRIC_TYPE, STEP, STEPVAR, NEIGHBORS)
{
    Schedule for STEP cycles and find the Best Freq Configuration (BFC)
    do
        if (BFC not set)   /* If first run */
           NEIGHBORS_SET = all frequency configurations
        else
           NEIGHBORS_SET = neighbors of BFC /*up to NEIGHBORS per cluster*/
        for FCONF in NEIGHBORS_SET
           /* Partially schedule the ready instructions of FCONF frequency
                configuration for STEP cycles, optimizing METRIC_TYPE */
           SCORE = cluster_and_schedule (METRIC_TYPE, STEP, FCONF)
           Store the scheduler's calculated SCORE into  SCORECARD [FCONF]
        Decrement STEP by STEPVAR until 1. /* Variable steps (optional) */
        BFC = Best Freq Configuration of SCORECARD, clear SCORECARD
    while there are unscheduled instructions in active set
    return BFC and scheduled code of BFC
}
```

```
/* In1: METRIC_TYPE: The metric type that the scheduler will optimize for.
   In2: STEP: Num of instrs to schedule before switching to next freq. conf.
   In3: FCONF: The architecture’s current frequency configuration.
   Out: Scheduled Code and metric value. */
cluster_and_schedule (METRIC_TYPE, STEP, FCONF)
{
   /* Restore ready list for this frequency configuration */
   READY_LIST = READY_LIST_ARRAY [FCONF]
   /* Restore current cycle. CYCLE is the scheduler’s internal cycle. */
   CYCLE = LAST_CYCLE [FCONF]
   Restore the Reservation Table state that corresponds to FCONF
   while (instructions left to schedule && STEP > 0)
      update READY_LIST with ready to issue at CYCLE, include deferred
      sort READY_LIST based on list-scheduling priorities
      while (READY_LIST not empty)
         select INSN, the highest priority instruction from the READY_LIST
         create LIST_OF_CLUSTERS[] that INSN can be scheduled at on CYCLE
         BEST_CLUSTER=best of LIST_OF_CLUSTERS[] by comparing for each cluster
               ↪calculate_heuristic(METRIC_TYPE,CLUSTER,FCONF,INSN,IPCL[])
         /* Try scheduling INSN on the best cluster */
         if (INSN can be scheduled on BEST_CLUSTER at CYCLE)
             schedule INSN, occupy LATENCY[FCONF][BEST_CLUSTER][INSN] slots
             IPCL [CLUSTER] ++  /* count number of instructions per cluster */
             remove INSN from READY_LIST
         /* If failed to schedule INSN on best cluster, defer to next cycle */
         if (INSN unscheduled)
             remove INSN from READY_LIST and re-insert it at CYCLE + 1
      /* No instructions left in ready list for CYCLE, then CYCLE ++ */
      CYCLE ++
      /* If we have scheduled for STEP cycles, finalize and exit */
      if (CYCLE $>$ LAST_CYCLE[FCONF] + STEP)
          Update READY_LIST_ARRAY[], LAST_CYCLE[] and Reservation Table
          return
}
```

```
/* In1: METRIC_TYPE: The metric type that the scheduler will optimize for.
   In2: CLUSTER: The cluster that INSN will be tested on.
   In3: FCONF: The architecture's current frequency configuration.
   In4: INSN: The instruction currently under consideration.
   In5: IPCL: The Instruction count Per CLuster (for dyn energy).
   Out: metric value of METRIC_TYPE if INSN scheduled on CLUSTER under FCONF*/
calculate_heuristic (METRIC_TYPE, CLUSTER, FCONF, INSN, IPCL[])
{
    START_CYCLE = earliest cycle INSN can be scheduled at on CLUSTER
    UCIFF_SC = START_CYCLE + LATENCY[FCONF][CLUSTER][INSN]
    switch (METRIC_TYPE)
        case ENERGY: return energy (CLUSTER, FCONF, UCIFF_SC, IPCL[])
        case EDP: return edp (CLUSTER, FCONF, UCIFF_SC, IPCL[])
        case ED2: return ed2 (CLUSTER, FCONF, UCIFF_SC, IPCL[])
        case DELAY: return UCIFF_SC
}
```