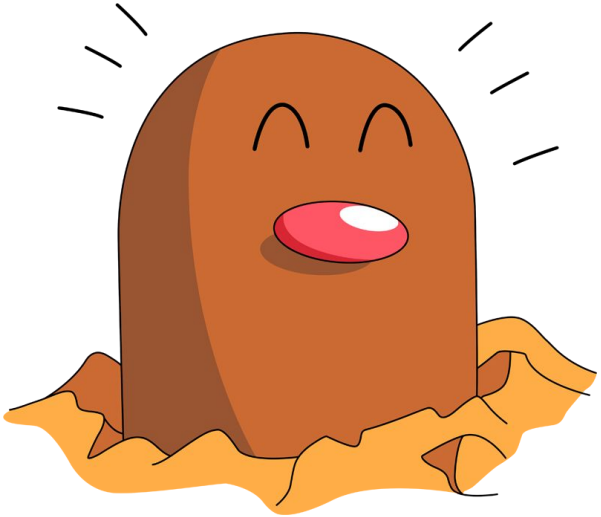
Fused and Composable Heterogeneous Cores

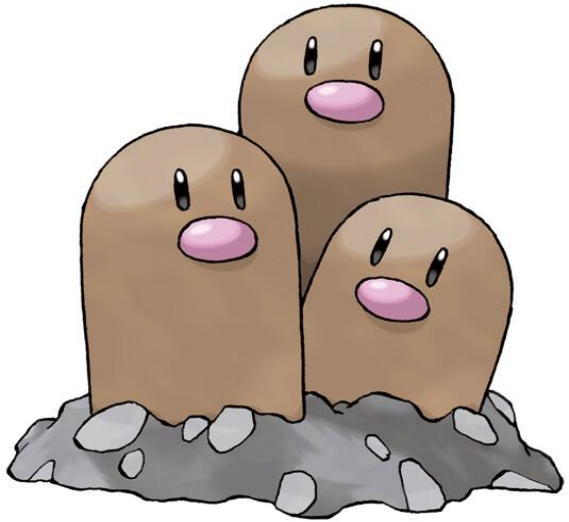
Roshan Nair and Anirudh Krishna Villivalam

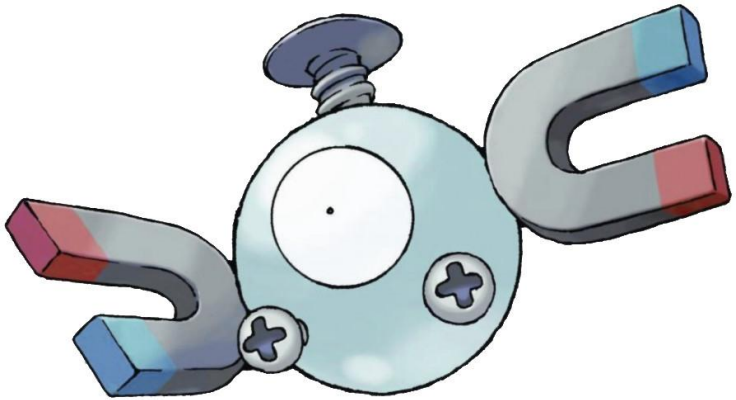
Single cores

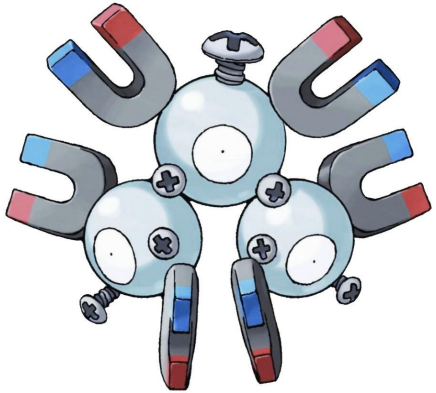


Evolution!!!

Fused/Composable cores







Core Fusion: Accommodating Software Diversity in Chip Multiprocessors

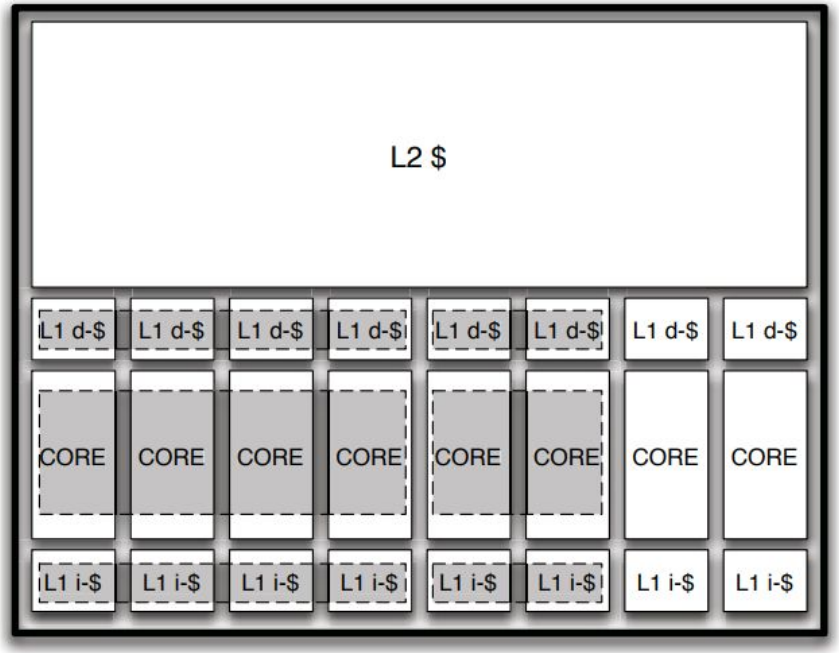
Motivation

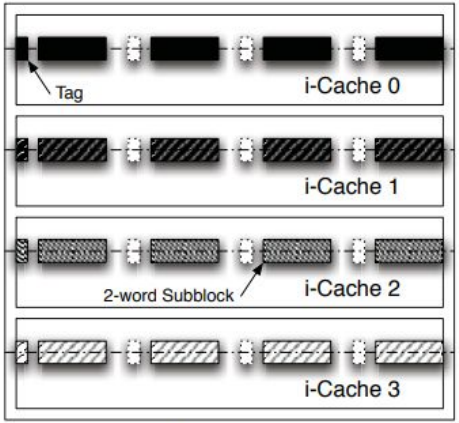
- Software Diversity and Evolution
  - Hardware can dynamically accommodate software’s parallel and sequential characteristics
- Homogenous
  - Design is singular oriented with each core being identical
- Parallelism is the Future
  - Software is changing to exploit more parallelism in algorithms and data structures
  - Hardware needs to be able to keep up with the expected performance of such optimizations
- Independence
  - Design bugs or hard faults in core may not necessarily affect the entire system

Contribution (Fused Core)

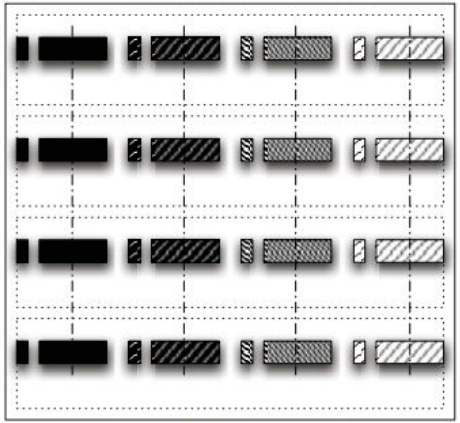
- Unit Core
  - Two-issue out of order
  - Private L1 instruction and data caches
  - Operate fully independently
- Fuse Core
  - Fuse unit cores into groups of 2 or 4
  - Effectively doubling or quadrupling issue width and hardware resources available
  - Multiple small cores -> one big core
- On-chip L2 Cache and Memory Controller

Contribution (Fused Core)





(a) Independent

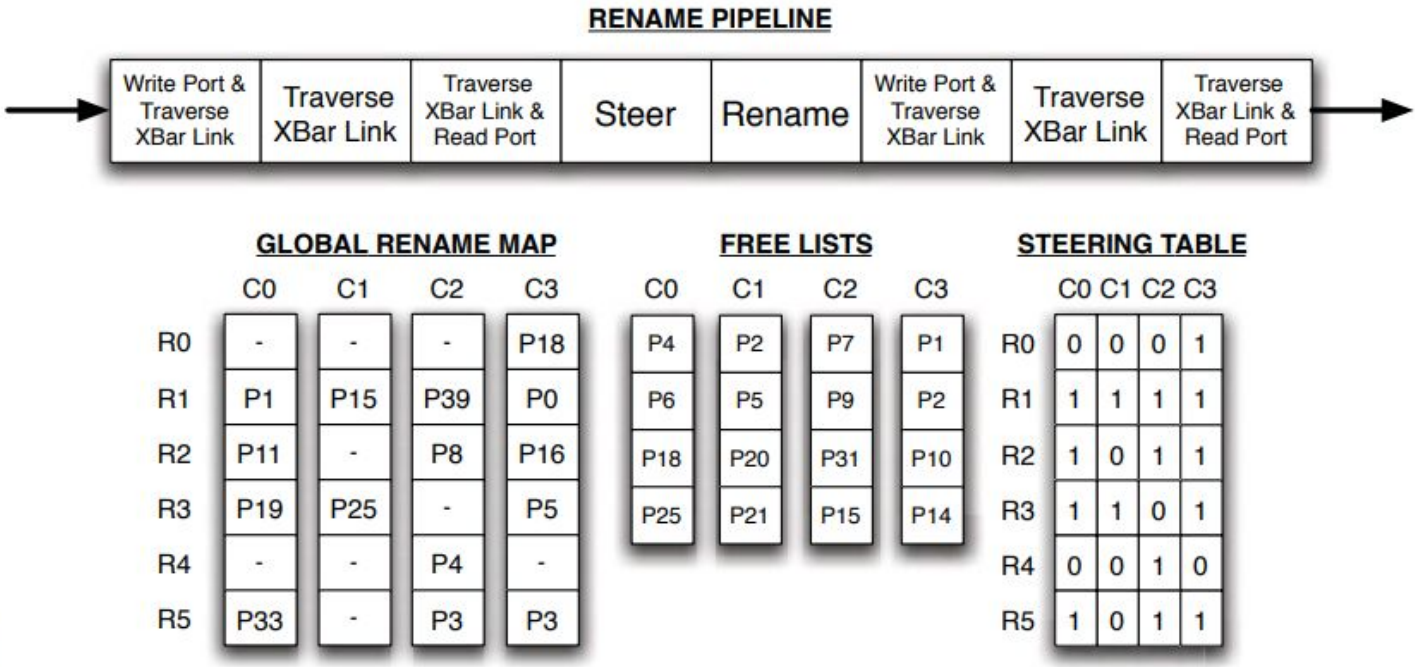


(b) Fused

Contribution (Front End)

- FMU (Fetch Management Unit)
  - 2 cycle latency from core to core (through FMU)
  - Fetches are aligned with core zero having the older instructions
    - Core zero will realign to maintain this invariant
  - I-cache holds replicas of tag depending on fusion mode
- Prediction
  - FMU gives priority based on different PC’s received from each core
- SMU (Steer Management Unit)
  - Steering table : map of arch registers to core
  - Free lists
  - Rename maps

Contribution (Front End)



Contribution (Back End)

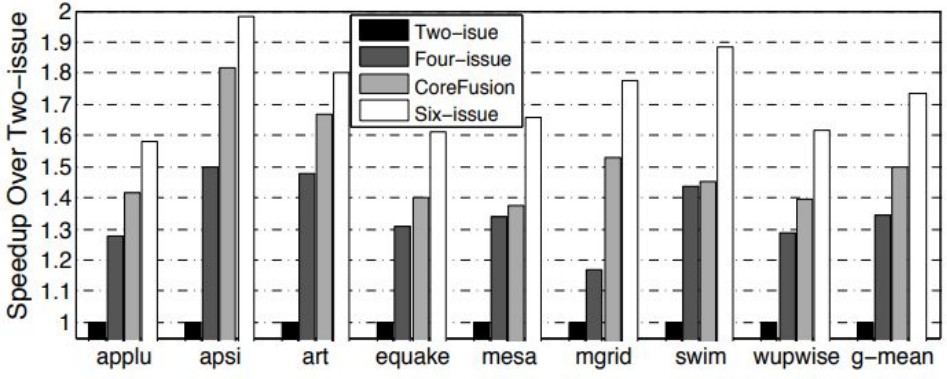
- Operand Crossbar
  - Copy instructions are stored in separate queue and wait till operands are ready
- ROB
  - When fused all 4 ROBs need to communicate
  - Need to maintain lockstep and may inject NOPs to force alignment
  - When stalled, other ROBs need to wait as well
  - Latency in signals handled by having “pre-commit” structures
- LSQ (Load Store Queue)
  - Use effective address bits to obtain which core and index
  - Implement a bank prediction to steer stores to correct core

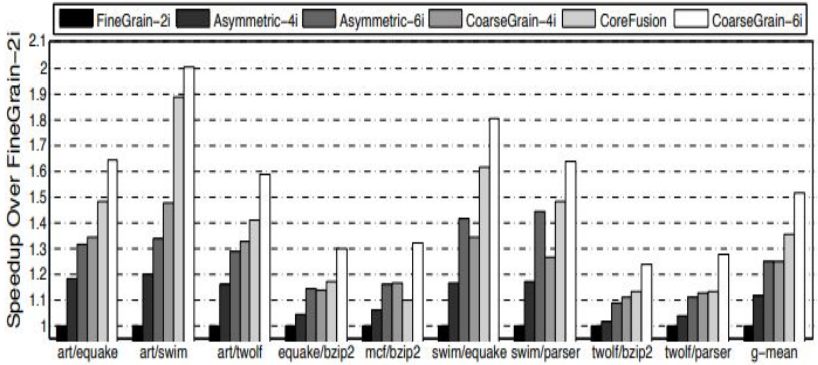
Contribution (ISA)

- FUSE
  - Fuse cores together for upcoming sequential operation
  - Instructions and i-cache are flushed
  - FMU, SMU, and i-cache are reconfigured
  - No change to d-cache (inherent coherence)
  - If can’t fuse -> don’t
- SPLIT
  - Split cores for upcoming parallel portion
  - Drain in flight instructions, then reconfigure data structures
  - Free for OS to re-allocate after this point

Merits

- How well is it able to balance TLP and ILP
  - Fused does better on ILP
  - Many cores do better with TLP
- Overall fused core performs ‘close’ to the better configuration
  - Usually an existing configuration does better than CoreFusion in one category
  - However in the opposite category, that same configuration does worse
  - Fused core can do both ‘relatively’ well





Failings

- Performance Factors
  - Not affected a lot by FMU delay
  - Restricted SMU bandwidth has around 3% impact
  - 18% from communication delays
  - NOPs and dummies in LSQ and ROB

Overall Conclusion

- Very novel and interesting approach
  - Fused core design lies in the domain of hardware “reconfigurability”
- Relatively easy to integrate
  - No software structure changes
  - Two ISA instructions added
  - Allows performance scalability as software grows over time
- Not perfect
  - Not able to beat performance of architectures designed for the extreme cases

Composable, Lightweight Processors

Motivation

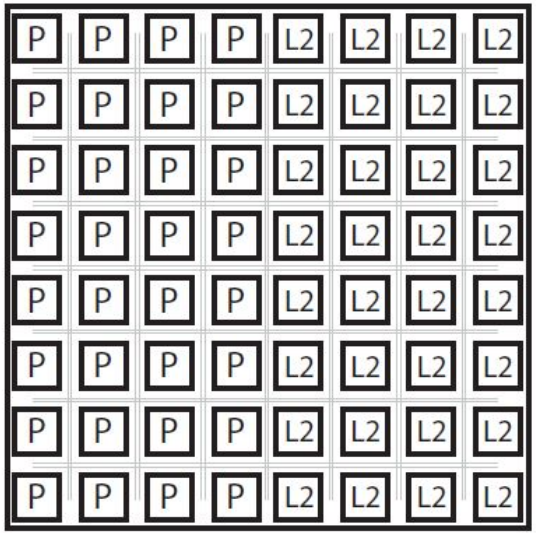
- Hardware designs are fixed
  - Cannot optimize for both TLP and ILP
- Also homogenous
  - Each core is similar, simple and low-power
- Parallelism is the Future, but Serialization is Timeless
  - Design focuses on optimizing ILP, TLP as well as energy
  - Software decides processor “growth” or “shrinking” for optimization
- Scalability
  - Design does not need physical sharing of structures increasing scalability up to 64-wide issue

Contribution (TFlex)

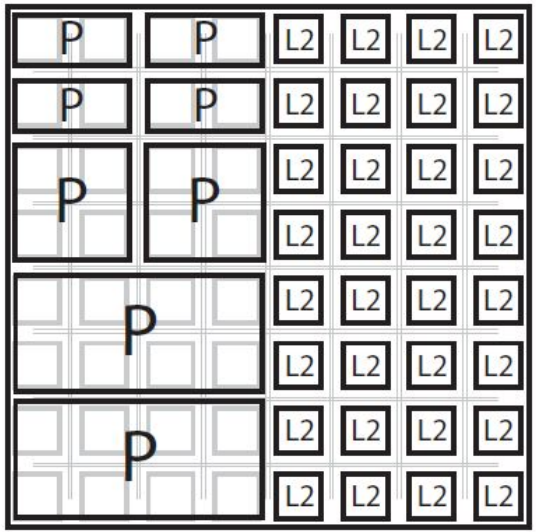
- Single Core (similar to CoreFusion)
  - Two-issue out of order
  - Private L1 instruction and data caches
  - Operate fully independently
- TFlex
  - Combine single cores into any number between 2 and 32 cores
  - Run-time software can optimize processor combination for ILP or TLP depending on number of threads
  - Multiple small cores -> work together as some big core. Structures not shared physically
- On-chip L2 Cache

Contribution (TFlex)

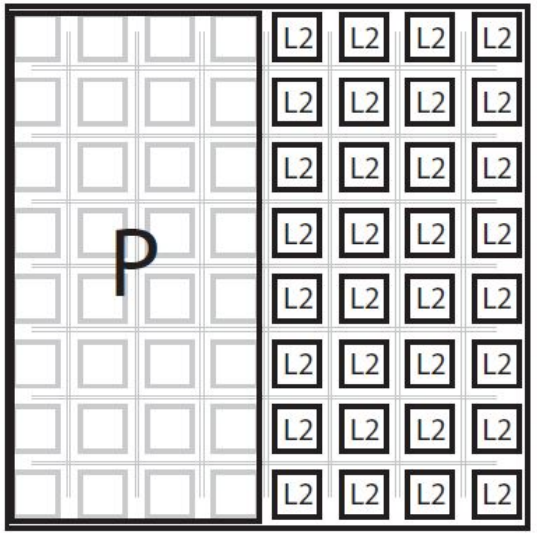
(a) 32 2-wide CLP config.



(b) 8-processor CLP config.



(c) One 64-wide CLP config.



Details of Instruction Set

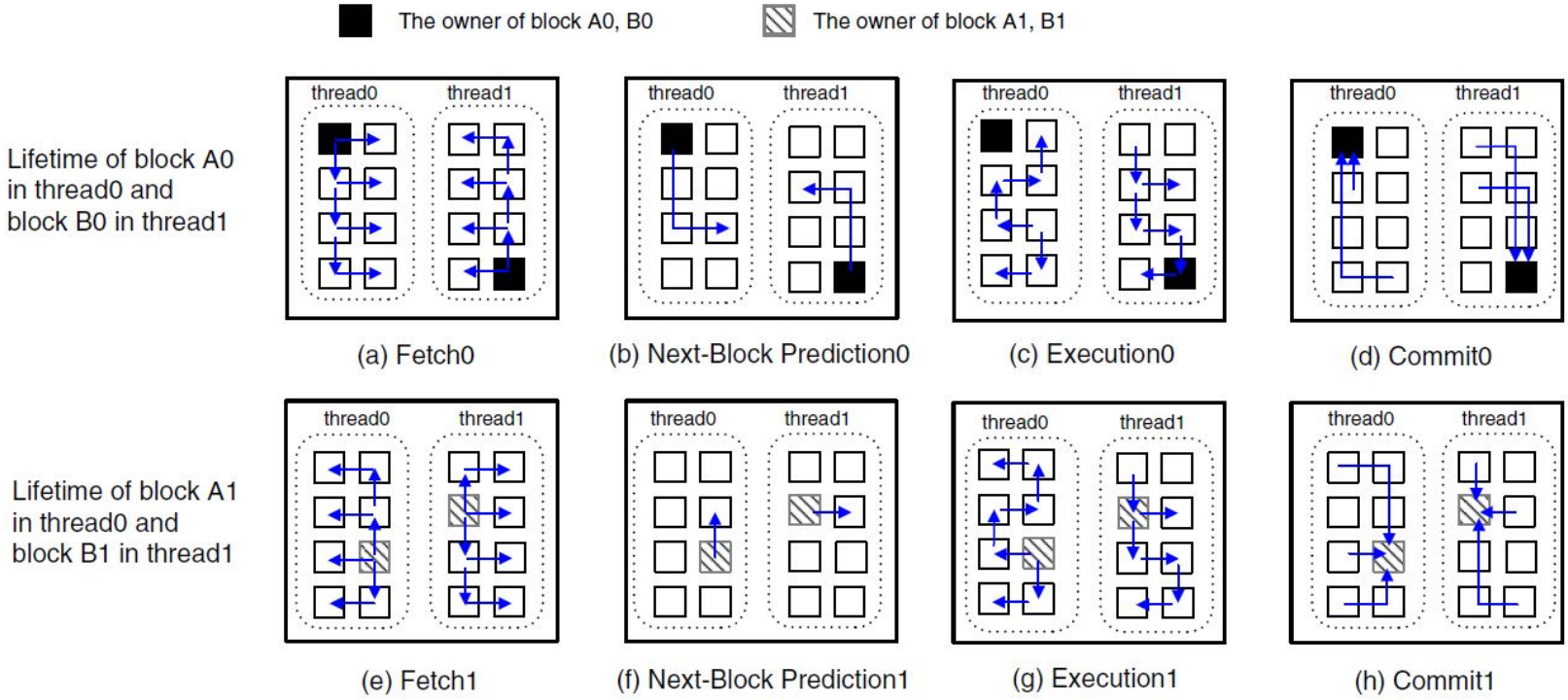
- EDGE ISA (from TRIPS)
  - Avoids distribution of each instruction by using Explicit Data Graph Execution
  - Instructions are encoded into sequence of atomic blocks
    - Control protocols act on large blocks (128 instructions) rather than each instruction
  - Encoding also replaces message broadcasting with point-to-point communication

Details of Microarchitectural structures

- Microarchitecural structures can vary linearly
  - Doubling cores -> doubling Load/Store queues, usable state in branch predictors, cache
  - Structures partitioned by address -> avoids physical centralization
    - Improves on limitations of TRIPS caused due to centralization
- Three hash functions used
  - Block starting address partitioned based on virtual address
    - Virtual address corresponds to PC
  - Instructions are given IDs in order and are interleaved
  - Data address partitioned based on data address with register interleaving

TFlex Operation - An Overview

- Blocks are assigned to “Owner Cores”
  - Responsible for fetching block and predicting next block
  - Forwards next block address to corresponding owner
  - Also performs flushing, detects block completion and committing


The owner of block A0, B0
The owner of block A1, B1
Lifetime of block A0 in thread0 and block B0 in thread1
Lifetime of block A1 in thread0 and block B1 in thread1
thread0
thread1
(a) Fetch0
(b) Next-Block Prediction0
(c) Execution0
(d) Commit0
(e) Fetch1
(f) Next-Block Prediction1
(g) Execution1
(h) Commit1

Merits

- Design eliminates need for physical sharing, broadcasting and reconfiguration
  - Increases scalability as well as allows for wider range of composing cores
- Control flow is easier due to nature of EDGE ISA
- Cores need not “combine” or “split” on a physical level
  - No latency for changing mode like in Core Fusion
- Design provides reasonable performance for both serial and parallel execution
  - Similar to Core Fusion, can perform relatively well for both cases

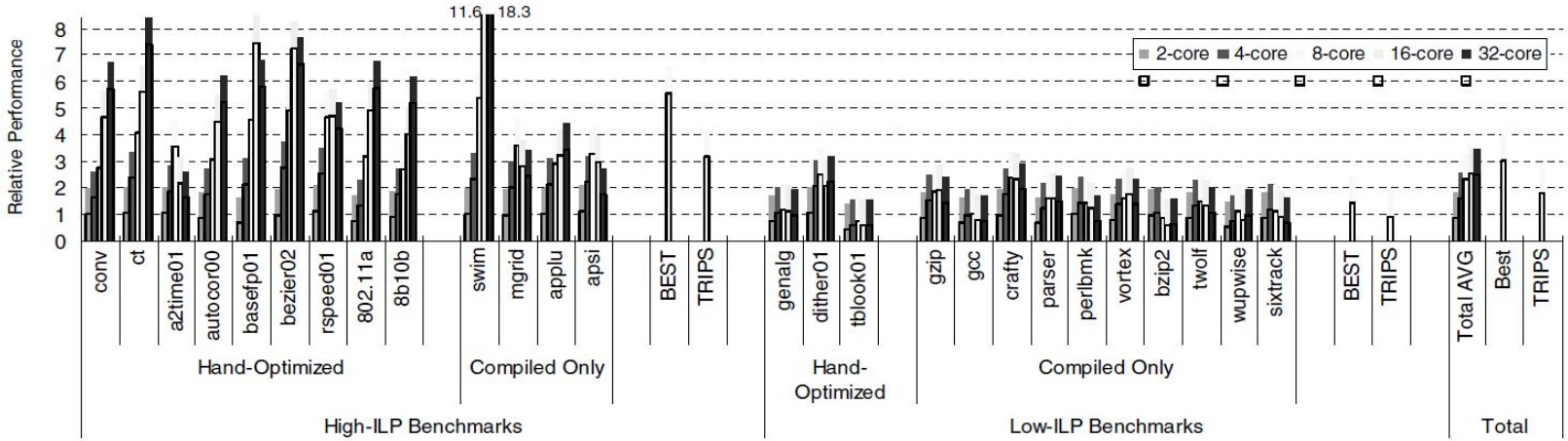


Figure 6: Performance of different applications running on 2 to 32 cores on a CLP normalized to a single TFlex core

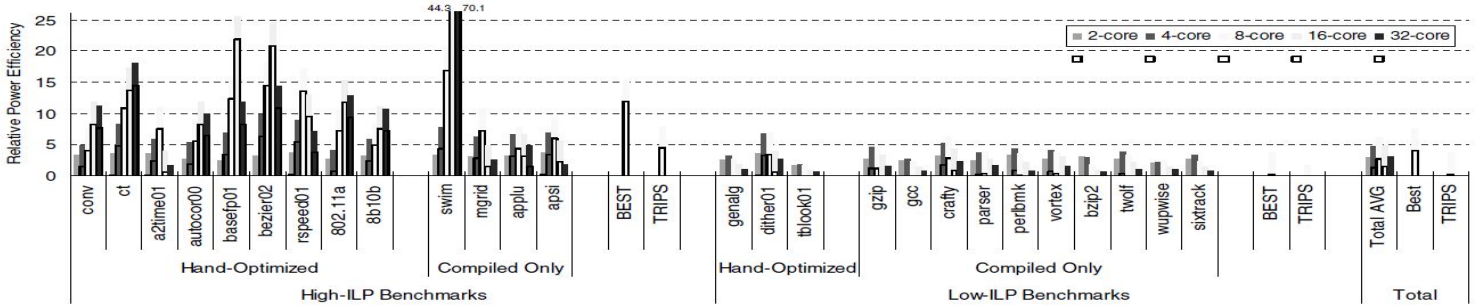


Figure 8: Performance^2/Watt normalized to a single TFlex core

Failings

- Mentions that they “envision multiple methods of controlling the allocation of cores to threads”
  - Ranges from OS monitoring to hardware structures
  - Vague and not very specific though this is a key design choice if this were to be implemented
- Relies on a non-standard EDGE ISA for distributed microarchitecture
  - Hard to integrate into industry
- Configuration relies on a lot of factors
  - Performance, area, or energy
  - In practice it is very hard to optimize one factor without considerable changes to another

Overall Conclusion

- Another interesting approach
  - Design relies on software to manage configuration
- Relatively lower hardware overhead
  - No duplication of structures needed
  - Does not need broadcast
- Choice of non-standard ISA might solve issues with standard ISAs
  - Transforming challenges into a different form which can be handled better