# Parallel Program Scheduling in AMPs

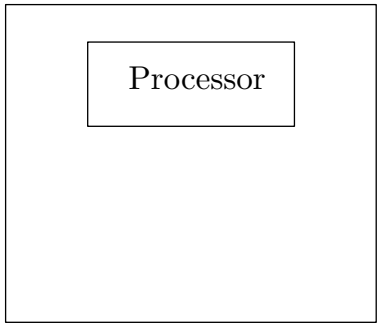
**Jyothi Krishna V S**
Guides: Shankar Balachandran and Rupesh Nasre

jkrishna@cse.iitm.ac.in

IIT Madras

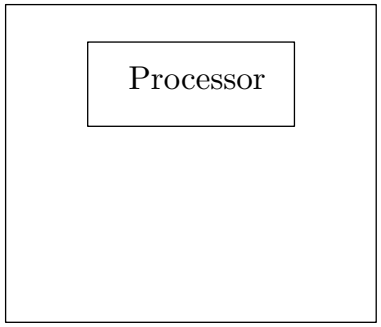
April 20, 2018

# CPU: Power Wall

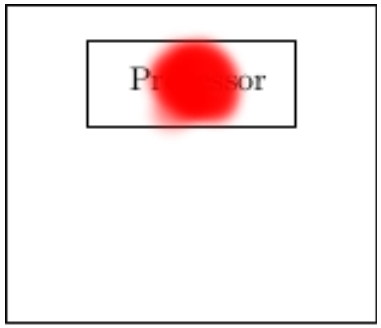



CPU

# CPU: Power Wall



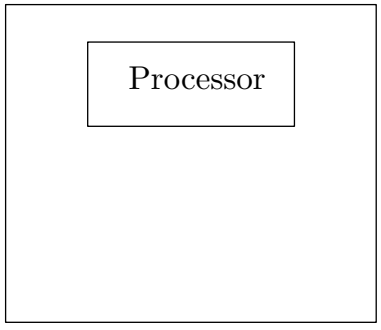

CPU

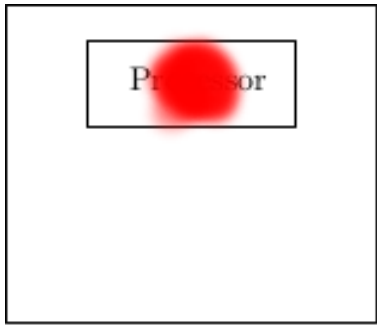


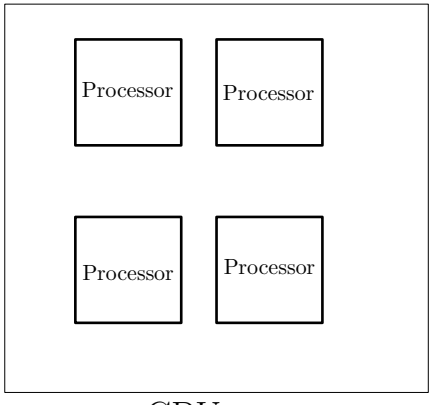
CPU

- Moore's Law hit the Power Wall
- Not practical to keep on increasing the frequency

# CPU: Power Wall

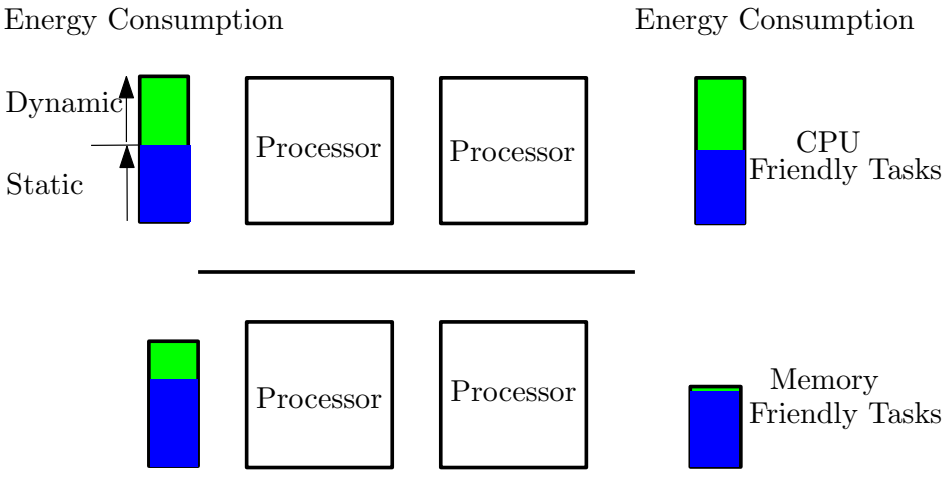



- Moore's Law hit the Power Wall
- Not practical to keep on increasing the frequency
- Move to Multicore environment, Multiple core working at lower frequency
- Move to parallel programming to keep on reaping benefit

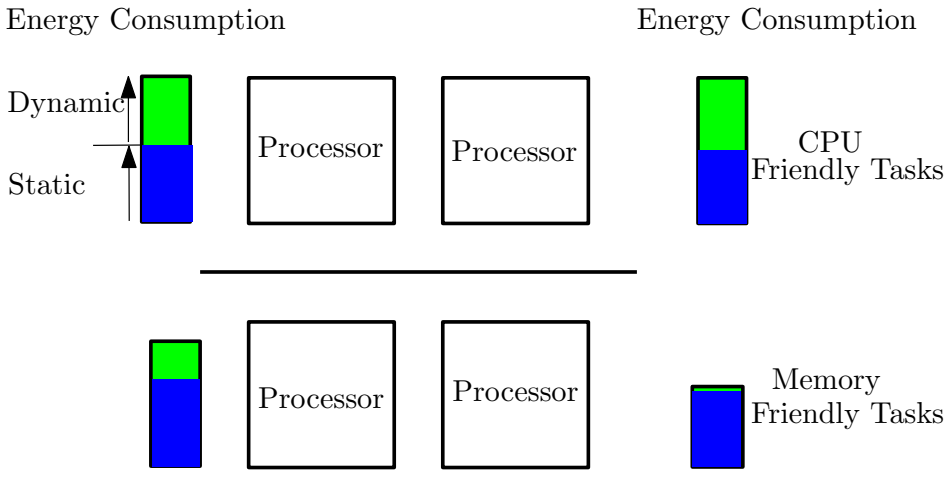
# High Static Cost of CPU

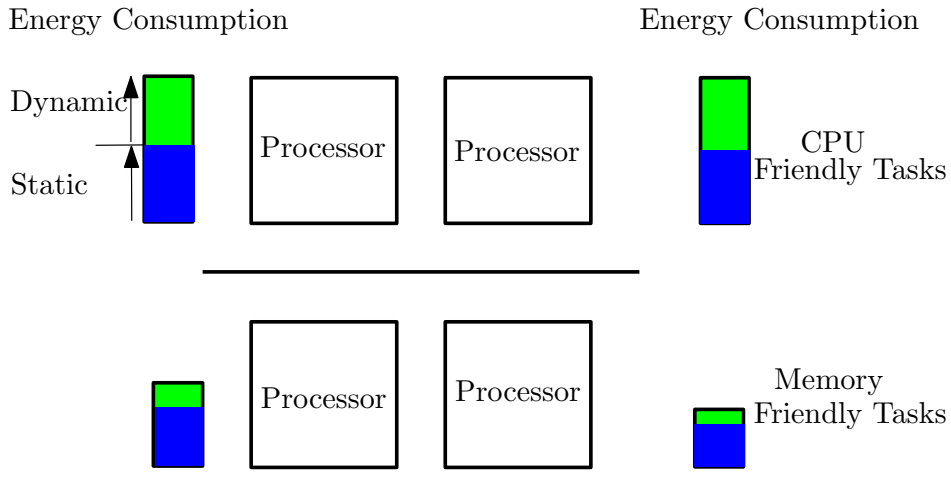



- CPU Friendly Tasks: Mostly ALU operation (active CPU cycles)
- Memory Friendly Tasks: Mostly memory/ Branch operations

# High Static Cost of CPU

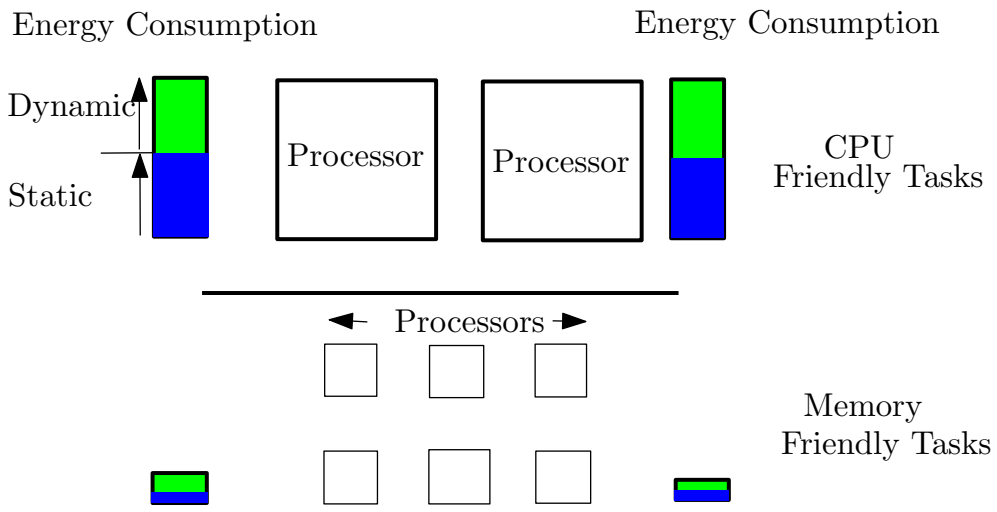
DVFS




- CPU Friendly Tasks: Mostly ALU operation (active CPU cycles)
- Memory Friendly Tasks: Mostly memory/ Branch operations
- DVFS: reduced the negative effect
- Still had high static cost to keep modules on.

# Asymmetric Multicore Processors (AMPs)




- CPU friendly tasks to big & powerful cores
- Memory friendly tasks to small & power efficient cores

eg. `big.LITTLE` from ARM, Tegra from NVIDIA

# AMP: Challenges with a multithreaded program

- What would be the optimal Hardware Configuration to run a Parallel Program?
  - Which type of core to run each thread on?
  - Does an AMP environment help? (Use one type of core at a time)
  - Number of resources of each type?
  - Optimal frequency configuration for each core type?

  CHOAMP, SIAM*

[1] * To submit

# AMP: Challenges with a multithreaded program

- What would be the optimal Hardware Configuration to run a Parallel Program?
  - Which type of core to run each thread on?
  - Does an AMP environment help? (Use one type of core at a time)
  - Number of resources of each type?
  - Optimal frequency configuration for each core type?

  CHOAMP, SIAM*
- Given a AMP environment what would be the optimal scheduling for a given parallel program?
  CES, SIAM*

[1] * To submit

# AMP: Challenges with a multithreaded program

- What would be the optimal Hardware Configuration to run a Parallel Program?
  - Which type of core to run each thread on?
  - Does an AMP environment help? (Use one type of core at a time)
  - Number of resources of each type?
  - Optimal frequency configuration for each core type?

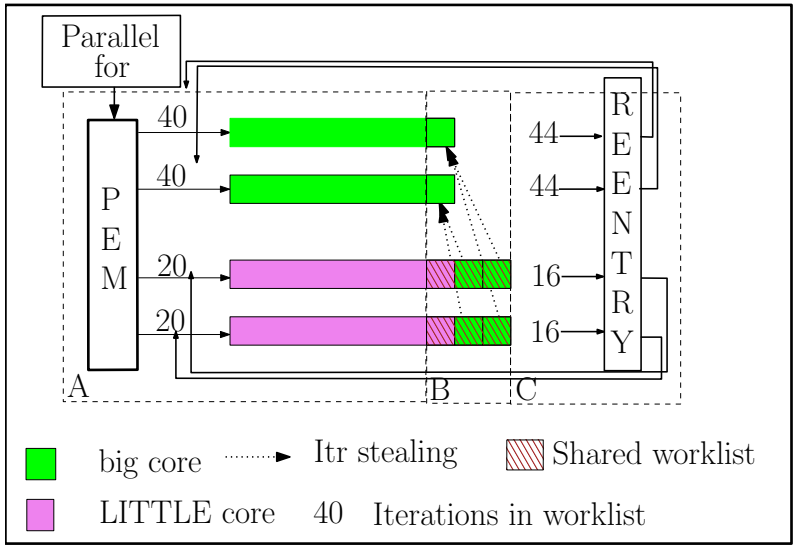
  CHOAMP, SIAM*
- Given a AMP environment what would be the optimal scheduling for a given parallel program?
  CES, SIAM*
- How to maintain fairness among parallel threads?
- How to make efficiently manage the Memory System (Prefetching, Variable Cache sizes etc)?

[1]

[1] * To submit

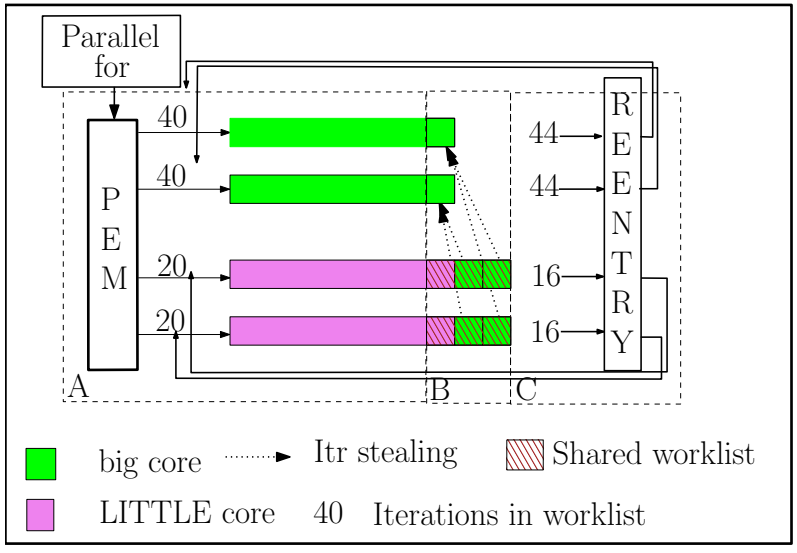
# Compiler Enhanced Scheduling (CES): Recap

- Uniform Parallel Workload
- Online Performance Evaluation Model(PEM)

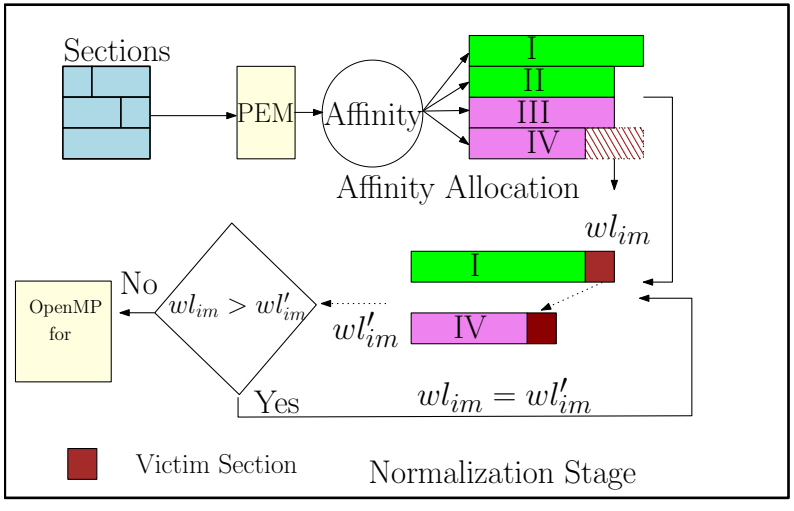


# Compiler Enhanced Scheduling (CES): Recap

- Uniform Parallel Workload
- Online Performance Evaluation Model(PEM)




- Non uniform Parallel Workload
- Offline Affinity and Normalization Scheduling



# CHOAMP: Cost Based Hardware Optimization for Asymmetric Multicore Processors

# Optimal Hardware

Given an input program, a user selected cost function and the possible hardware configurations, can we find an optimal hardware configuration which can run the program with minimal cost.

- Static/Compile time: Chen and John [2]
- Dynamic: Execution time decisions : HMP Scheduling[3]
- Hybrid: Compile Time Instrumentation and Runtime Decision: PIE[4], CES

---

[2] J. Chen and L. K. John. Efficient program scheduling for heterogeneous multi-core processors. in DAC '09

[3] http://www.arm.com

[4] K. Van Craeynest et al. Scheduling Heterogeneous Multi-cores Through Performance Impact Estimation (PIE), ISCA '12

# Static v/s Dynamic Scheduling

- Static
  - Unknown variables
  - Unknown Thread-to-Core Mapping
  - Hardware Resource Reservation
  - Scalable over large hardware configurations

# Static v/s Dynamic Scheduling

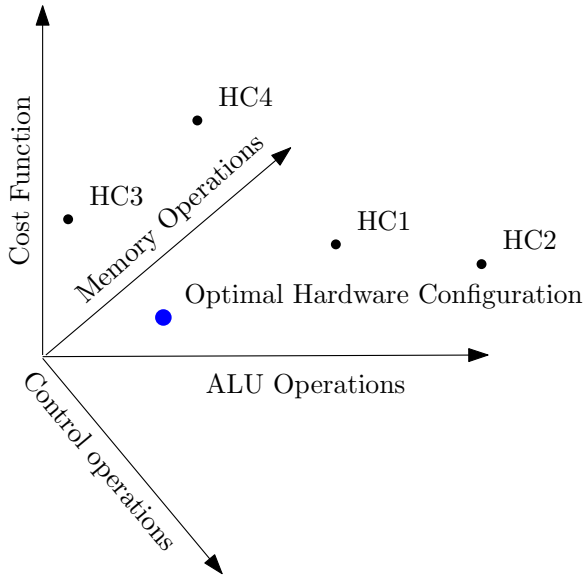
- Static
  - Unknown variables
  - Unknown Thread-to-Core Mapping
  - Hardware Resource Reservation
  - Scalable over large hardware configurations
- Dynamic
  - Runtime overhead
  - Scalability of the engine
  - Accurate knowledge of workload

# Static Scheduling

- Unknown Loop bounds: Estimate unknown variables based on known variables.
- Unknown wasted cycles: Estimate the expected delay by learning for large set of examples.

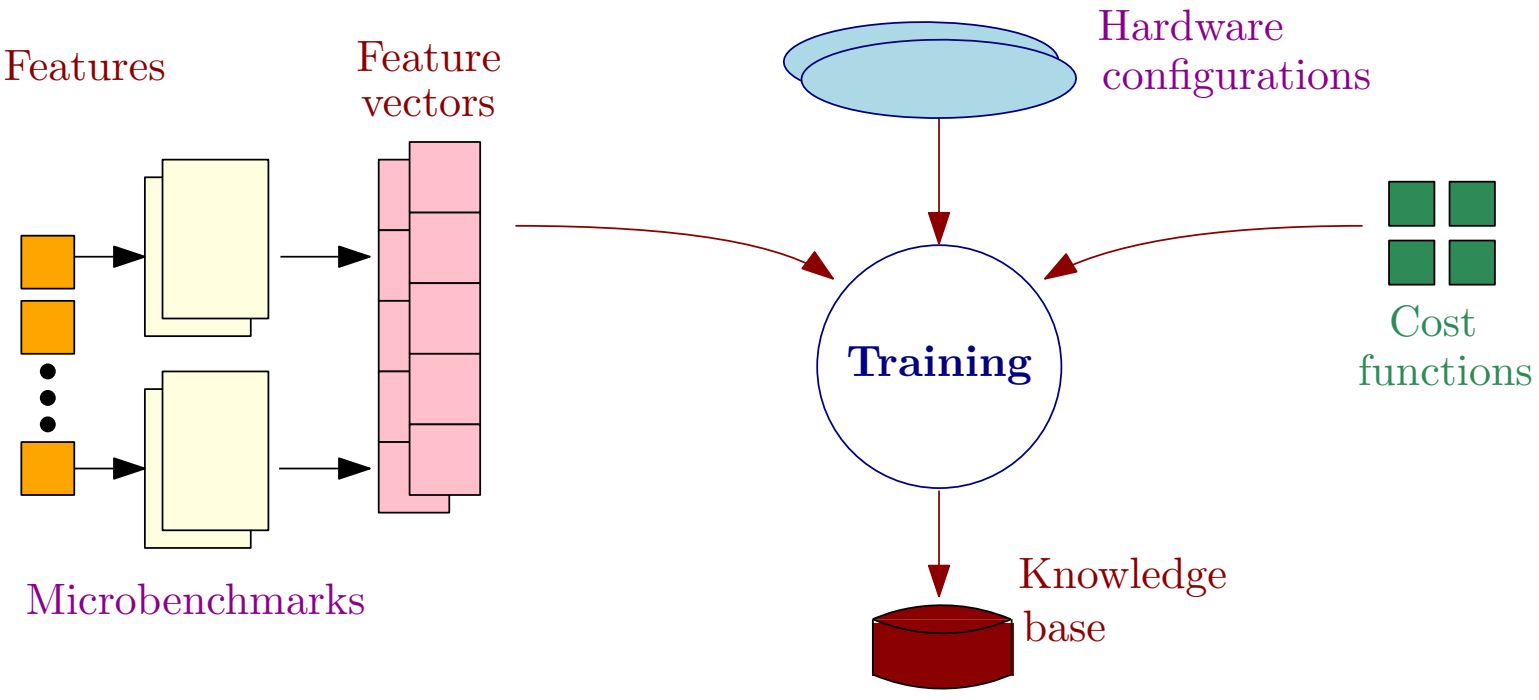
# Static Scheduling

- Unknown Loop bounds: Estimate unknown variables based on known variables.
- Unknown wasted cycles: Estimate the expected delay by learning for large set of examples.



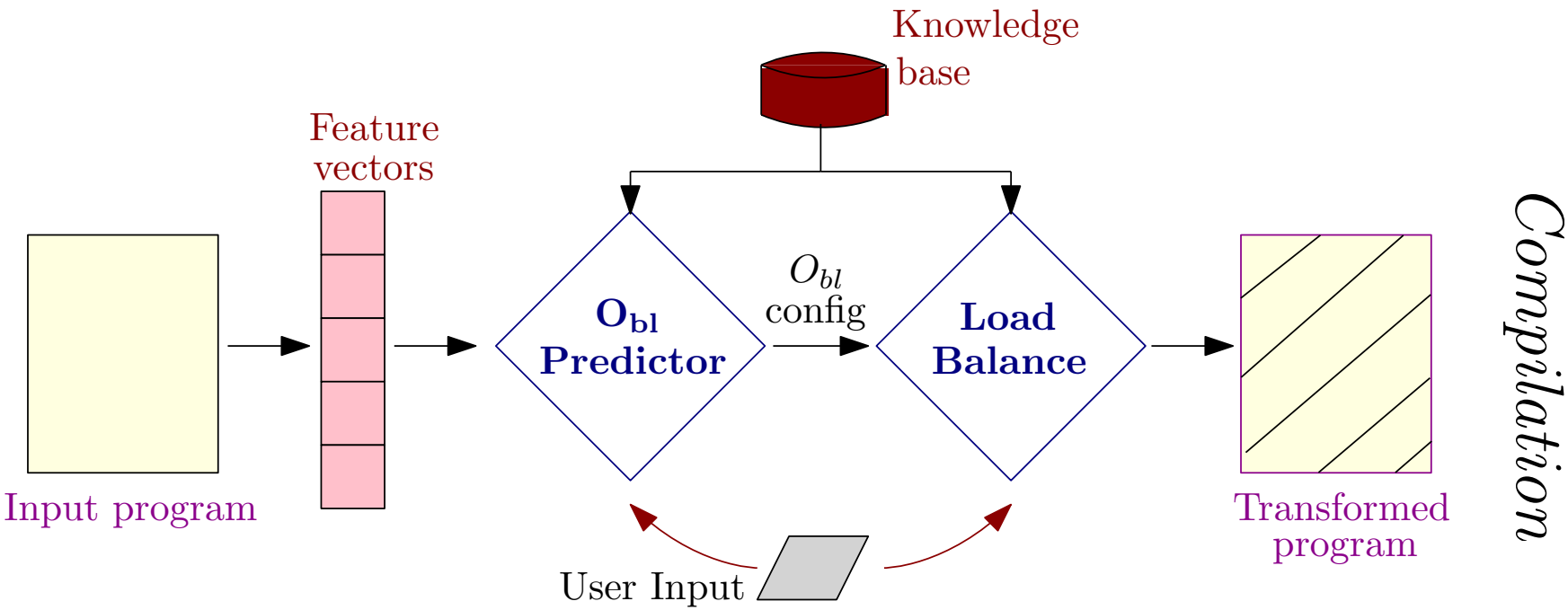

A `Predictor` trained with a set of program features (selected based on architectural differences in AMP cores and the choice of parallel program) and hardware configurations to identify the optimal hardware configuration

# CHOAMP: Training Phase

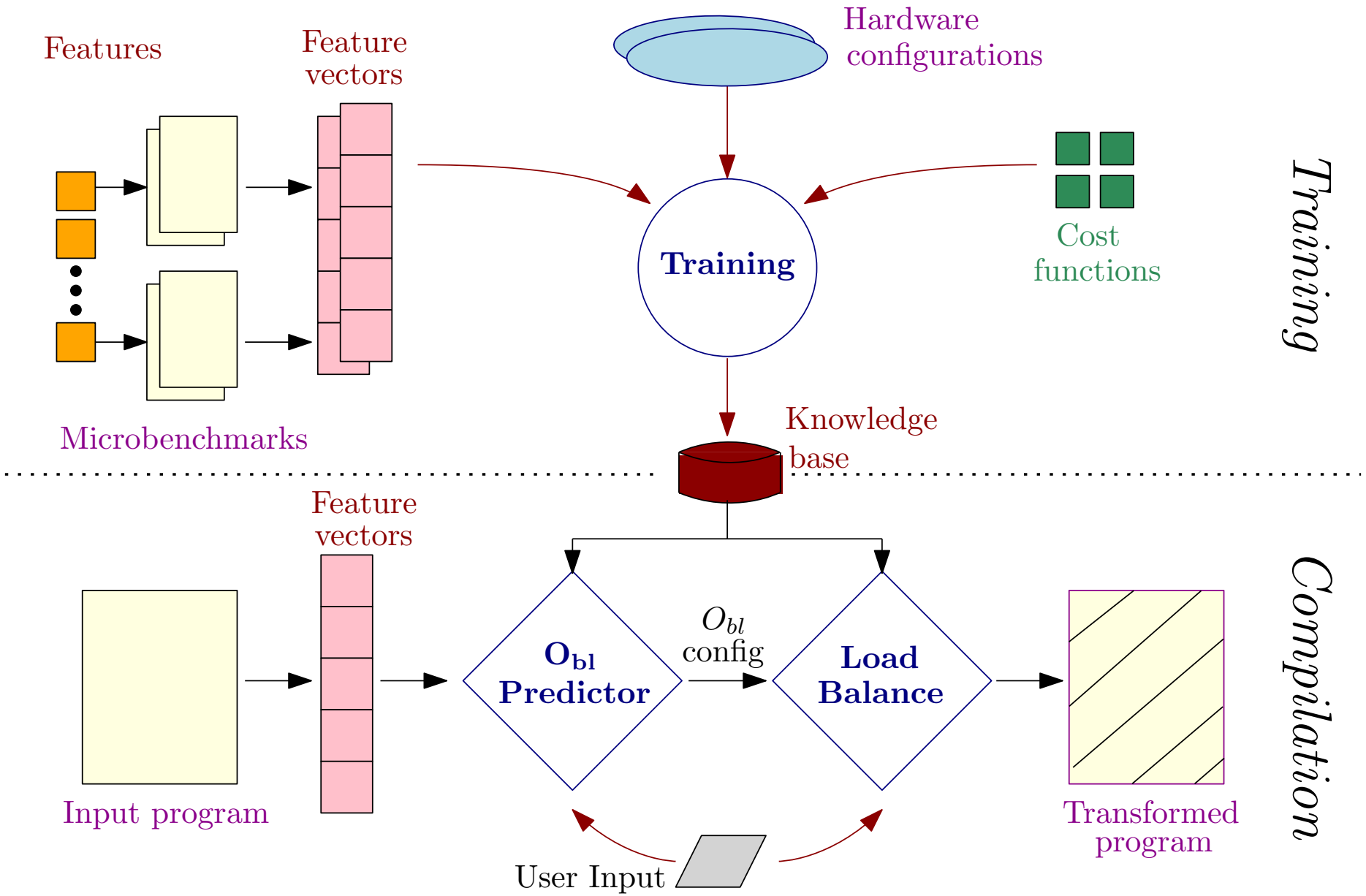



*Training*

# CHOAMP: Compilation Phase

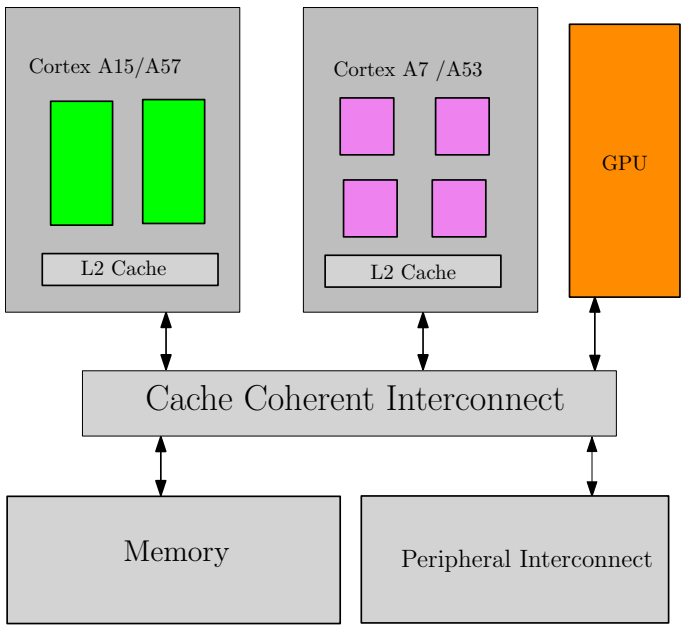


# CHOAMP: The Big Picture



# Test Environment : big.LITTLE

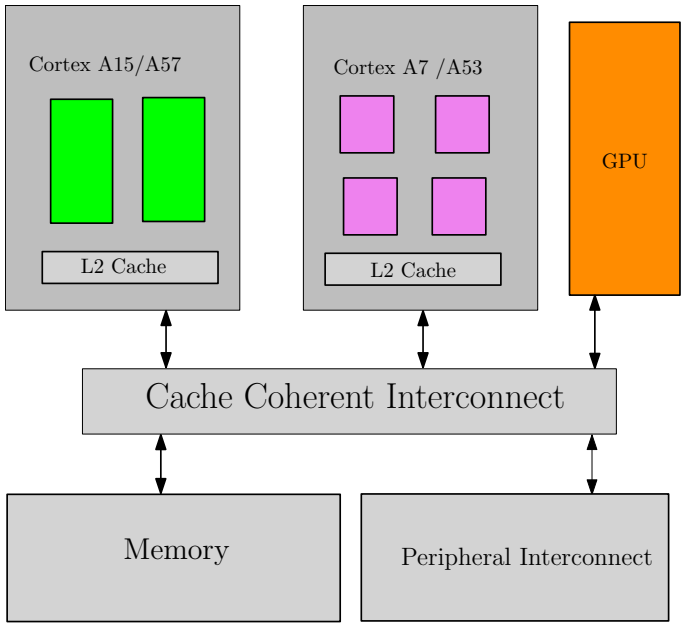
big.LITTLE System Design




| Core Types | Cortex-A7 | Cortex-A15 |
|---|---|---|
| Pipeline | simple 8 stage in-order | Out of Order, multi-issue |
| Frequency | 600 - 1400 MHz | 800 - 2000 MHz |
| Speed | 1.9 DMIPS | 4.01 DMIPS |
| Instruction Set | Thumb-2 | |

# Test Environment : big.LITTLE

big.LITTLE System Design



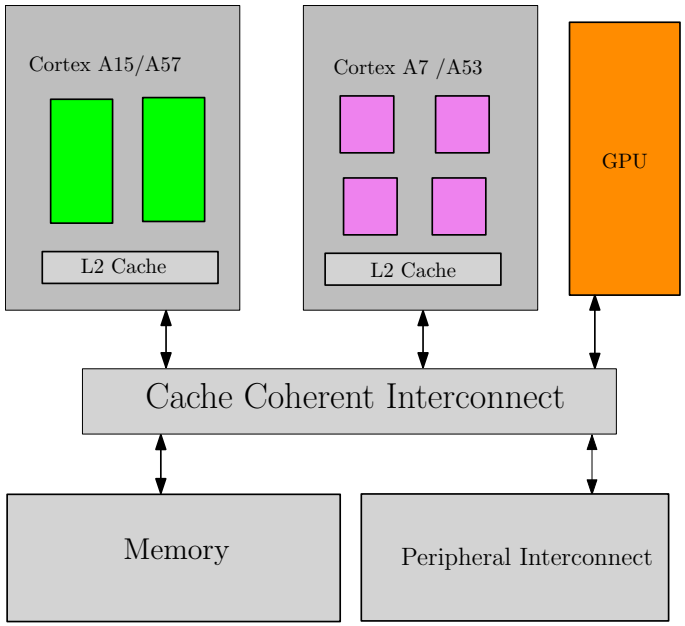

| Core Types | Cortex-A7 | Cortex-A15 |
|---|---|---|
| Pipeline | simple 8 stage in-order | Out of Order, multi-issue |
| Frequency | 600 - 1400 MHz | 800 - 2000 MHz |
| Speed | 1.9 DMIPS | 4.01 DMIPS |
| Instruction Set | Thumb-2 | |

| A15 Freq | Single Core Idle | Single Core Peak | Four Cores Idle | Four Cores Peak |
|---|---|---|---|---|
| 2.0 GHz | 0.95 | 2.40 | 1.15 | 5.28 |
| 1.8 GHz | 0.70 | 1.65 | 0.70 | 3.50 |
| **1.4 GHz** | **0.38** | **0.85** | **0.40** | **2.30** |
| 1.2 GHz | 0.30 | 0.70 | 0.32 | 1.80 |

| A7 Freq | Single Core Idle | Single Core Peak | Four Cores Idle | Four Cores Peak |
|---|---|---|---|---|
| 1.4 GHz | 0.20 | 0.50 | 0.22 | 1.40 |

# Test Environment : big.LITTLE

big.LITTLE System Design




| Core Types | Cortex-A7 | Cortex-A15 |
|---|---|---|
| Pipeline | simple 8 stage in-order | Out of Order, multi-issue |
| Frequency | 600 - 1400 MHz | 800 - 2000 MHz |
| Speed | 1.9 DMIPS | 4.01 DMIPS |
| Instruction Set | Thumb-2 | |

| A15 Freq | Single Core Idle | Single Core Peak | Four Cores Idle | Four Cores Peak |
|---|---|---|---|---|
| 2.0 GHz | 0.95 | 2.40 | 1.15 | 5.28 |
| 1.8 GHz | 0.70 | 1.65 | 0.70 | 3.50 |
| **1.4 GHz** | **0.38** | **0.85** | **0.40** | **2.30** |
| 1.2 GHz | 0.30 | 0.70 | 0.32 | 1.80 |

| A7 Freq | Single Core Idle | Single Core Peak | Four Cores Idle | Four Cores Peak |
|---|---|---|---|---|
| 1.4 GHz | 0.20 | 0.50 | 0.22 | 1.40 |

Parallel Program : OpenMP

# Feature Selection

- Low level : based on Hardware dissimilarities of `big` and `LITTLE`
  - ALU operations
  - Memory operations
  - Branch operations
  - False sharing

# Feature Selection

- Low level : based on Hardware dissimilarities of `big` and `LITTLE`
  - ALU operations
  - Memory operations
  - Branch operations
  - False sharing
- High Level: synchronization constructs provided by OpenMP:
  - Barriers
  - Atomics
  - Critical sections
  - Flush operations
  - Reduction operations

# Implementation

- Hardware Configurations: Odroid XU3 [5]
  - Core Configurations: 4L4b, 0L4b, 4L2b, 2L2b, 4L0b
  - big Frequencies : 2GHz, 1.8GHz, 1.6GHz, 1.4 GHz, 1.2GHz
- Micro-benchmark generation: Python 2.7
- Dynamic Scheduling: HMP and CES[6] for dynamic load balancing
- Analysis and Transformation: IMOP[7]
- Regression Tool: Java Scientific Library[8]
  - Regression Engines: Linear, Quadratic and Gaussian
- Benchmark : NPB[9]

[5] http://www.hardkernel.com/main/products/prdt_info.php?g_code=g140448267127

[6] J. K. Viswakaran Sreelatha, and S. Balachandran. 2016. Compiler Enhanced Scheduling for OpenMP for Heterogeneous Multiprocessors. EEHCO '16

[7] Nougrahiya and Nandivada, IMOP: http://www.cse.iitm.ac.in/~amannoug/imop/

[8] Flanagan M, http://www.ee.ucl.ac.uk/~mflanaga/java/index.html

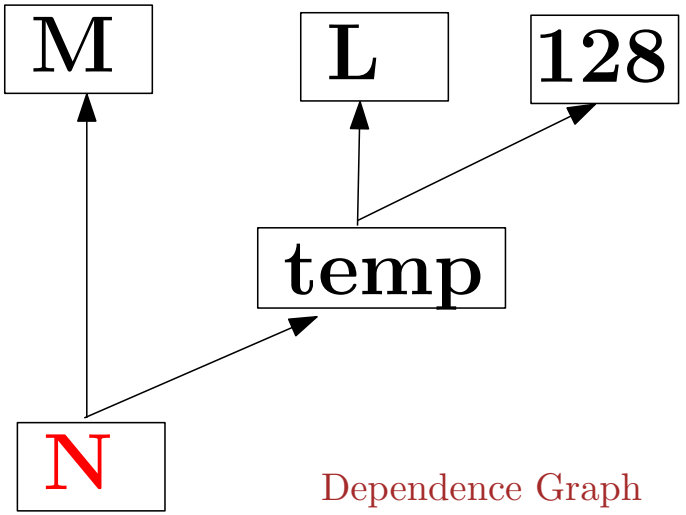
[9] Bailey et al. The NAS parallel benchmarks-summary and preliminary results

# Walk Through Example

```
#define M 50000
int f(int *s, int A[], int cumSum[], int L) {
     int MAX = 0, localSum = 0, temp = L/128;
     int N = M - temp; /* Unknown Variable*/
  #pragma omp parallel
  { int i, j;
    #pragma omp for reduction(+:localSum)
    for(i = 0; i<N ; i++) {
      localSum += A[i]; cumSum[i] = 0;
      for(j=0;j<N;j++) {
        if(j<=i)  cumSum[i] += A[j];
      }
      #pragma omp critical
      {  if(MAX < A[i]) MAX = A[i]; }
  }}
  return MAX;
}
```

# Walk Through Example

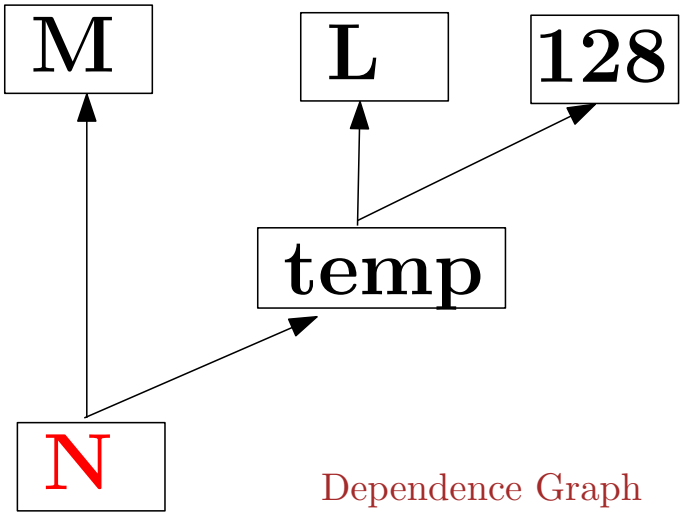
```
#define M 50000
int f(int *s, int A[], int cumSum[], int L) {
     int MAX = 0, localSum = 0, temp = L/128;
     int N = M - temp; /* Unknown Variable*/
  #pragma omp parallel
  { int i, j;
    #pragma omp for reduction(+:localSu
    for(i = 0; i<N ; i++) {
       localSum += A[i]; cumSum[i] =
       for(j=0;j<N;j++) {
          if(j<=i)  cumSum[i] += A[j
       }
       #pragma omp critical
       {  if(MAX < A[i]) MAX = A[i]; }
  }}
  return MAX;
}
```




Dependence Graph

# Walk Through Example

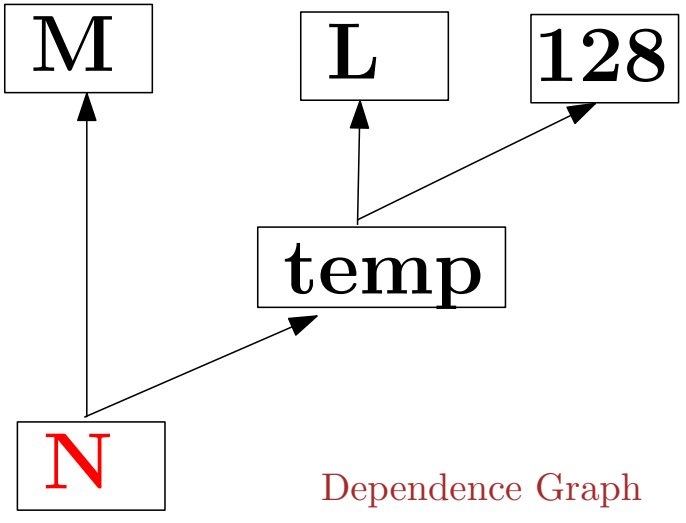
```
#define M 50000
int f(int *s, int A[], int cumSum[], int L) {
     int MAX = 0, localSum = 0, temp = L/128;
     int N = M - temp; /* Unknown Variable*/
  #pragma omp parallel
  { int i, j;
    #pragma omp for reduction(+:localSu
    for(i = 0; i<N ; i++) {
       localSum += A[i]; cumSum[i] =
       for(j=0;j<N;j++) {
          if(j<=i)  cumSum[i] += A[j
       }
       #pragma omp critical
       {  if(MAX < A[i]) MAX = A[i]; }
  }}
  return MAX;
}
```




Dependence Graph

# Walk Through Example

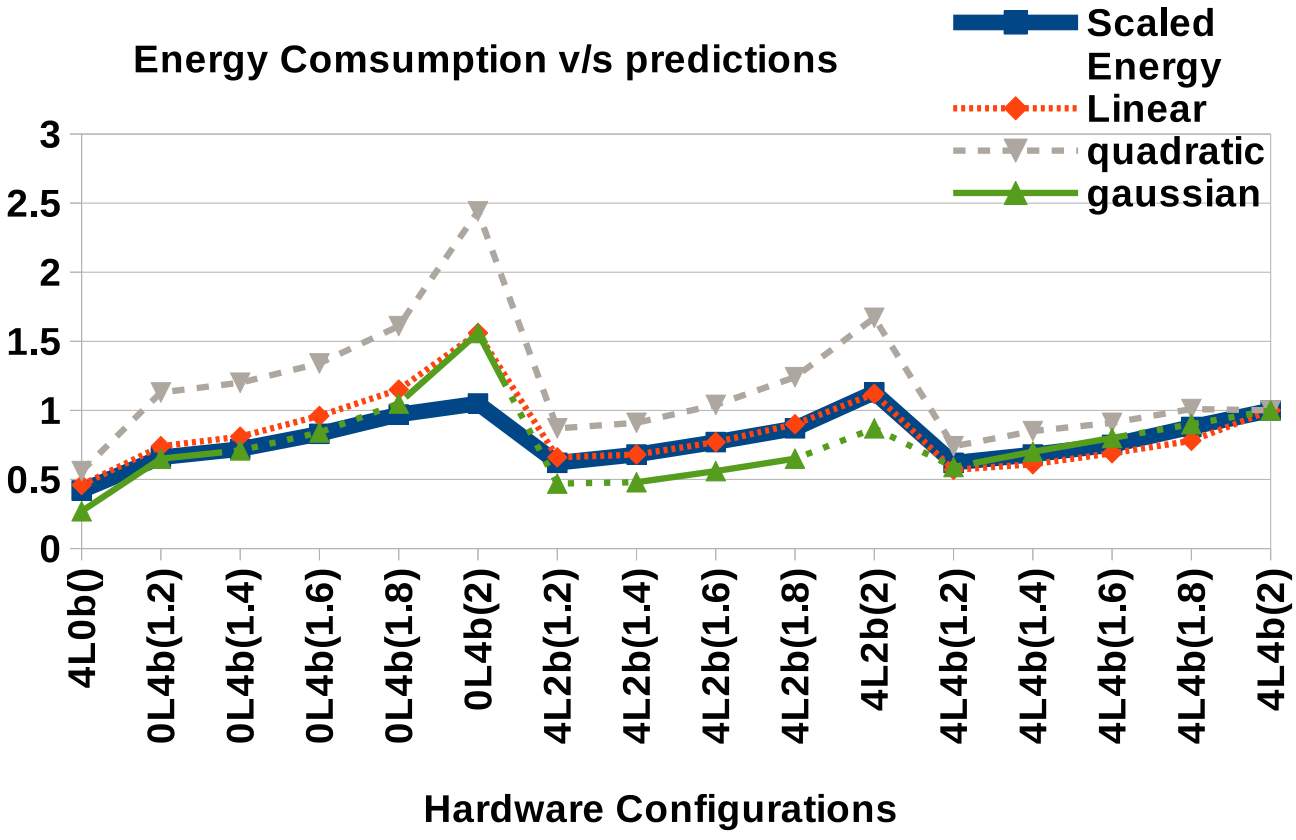
| Op | Result range |
|---|---|
| i + j | $range(MAX(R(i)) + MAX(R(j)))$ |
| i * j | $range(MAX(R(i)) * MAX(R(j)))$ |
| i - j | range(MAX($R(i)$) - MIN($R(j)$)) |
| i / j | range(MAX($R(i)$) / MIN($R(j)$)) |
| i >> j | i - MAX($R(j)$) |
| i % j | $R(j)$ |
| i = j | MAX($R(i)$, $R(j)$) |
| MAX($R(i)$): Known maximum value in range i<br>R(k): Range to which the value k belongs | |



Dependence Graph

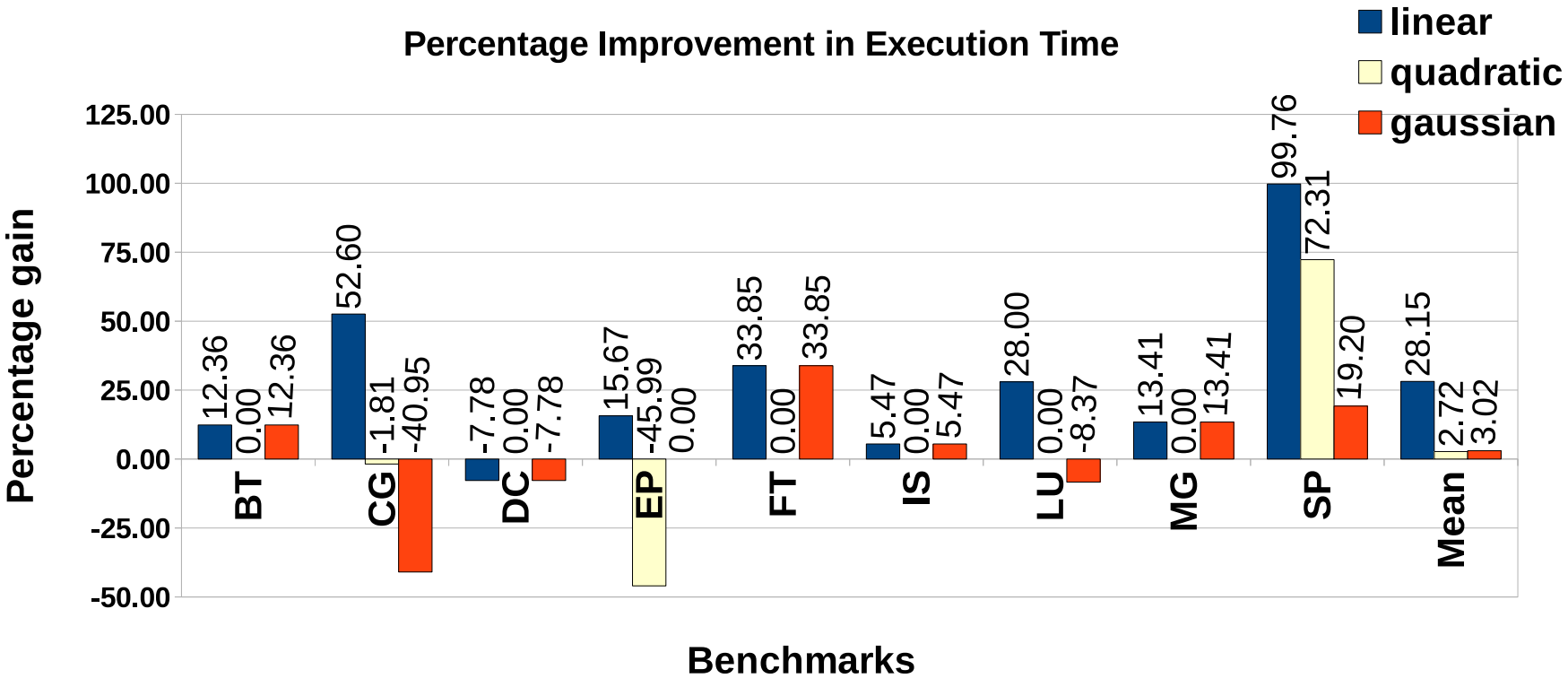
- L into largest non empty bucket.
- L $\epsilon$ $R_{4-5}$
- temp = 50000/128 = $R_{2-3}$
- N = 50000 - $R_{2-3}$ = $R_{4-5}$ = 50000

# Prediction v/s real world



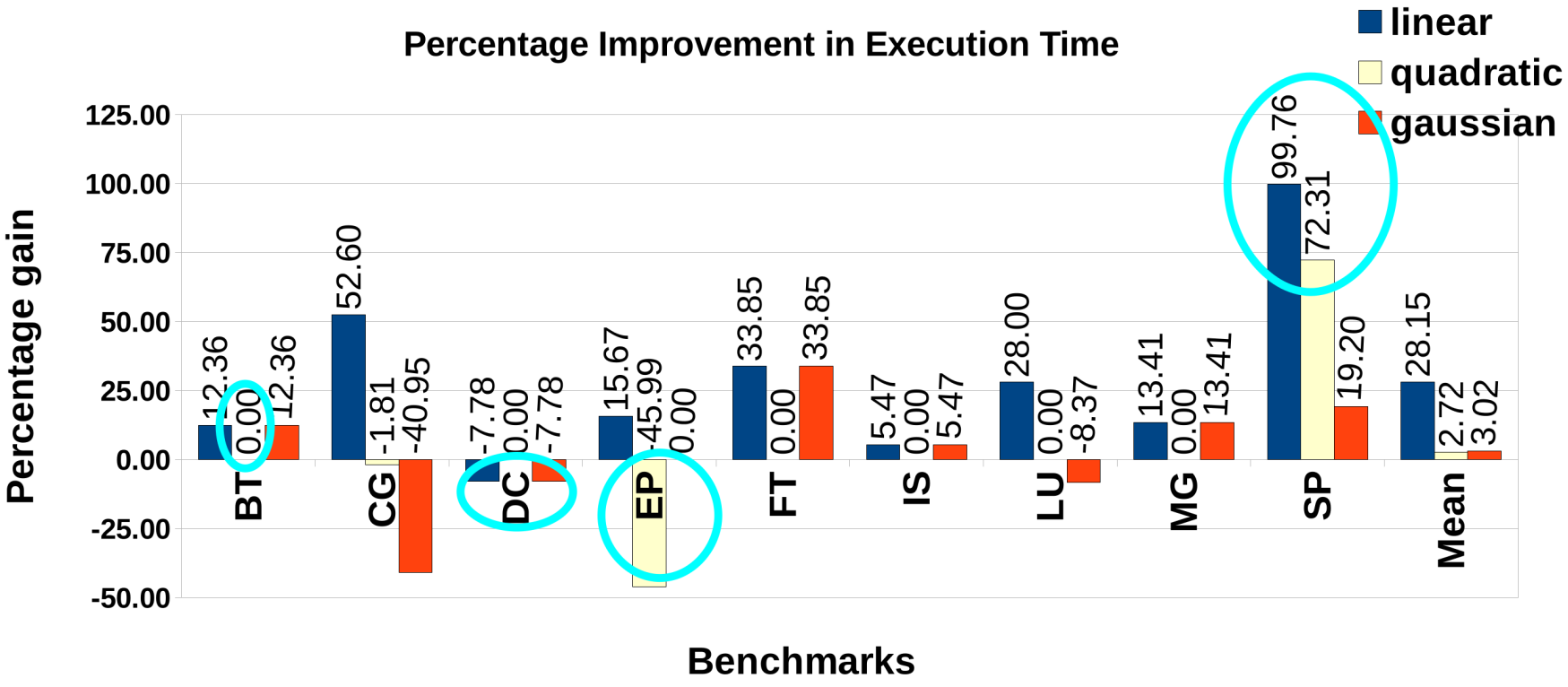

Scaled Prediction of Energy consumption by `Predictor` using different learning kernels for our example compared to actual (Bold Blue). Shows how good the `Predictor` is.

# CHOAMP: EXECUTION TIME



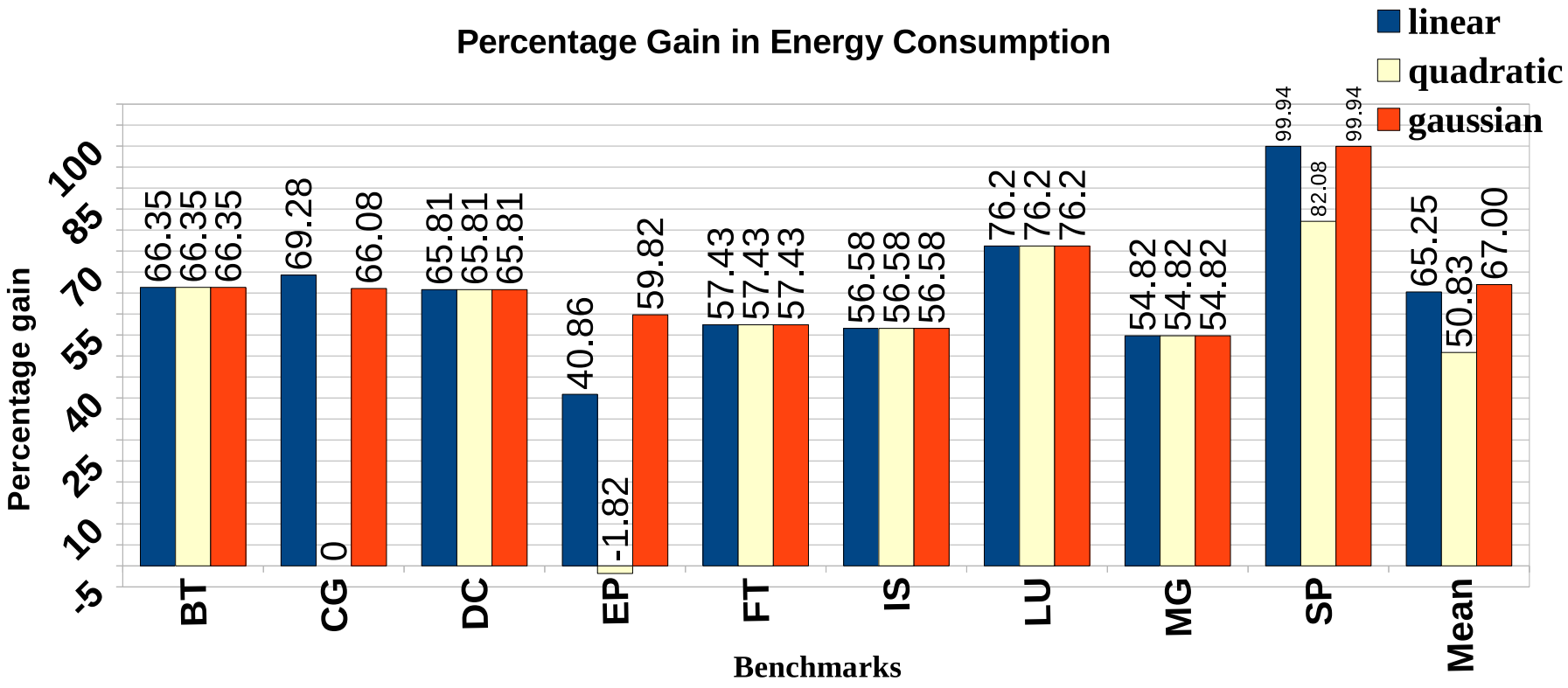

- On an average 28% improvement with Linear Regression Oracle

# CHOAMP: EXECUTION TIME



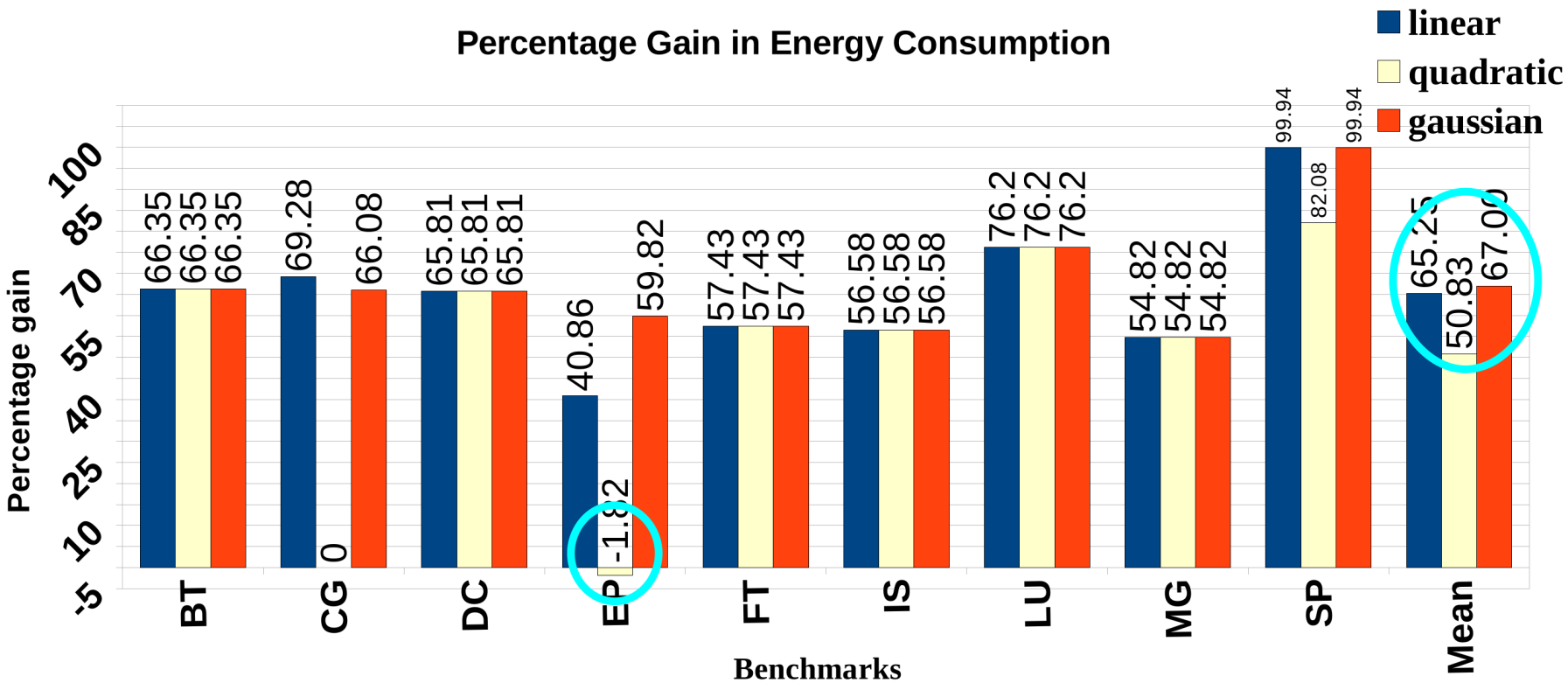

- BT: Selecting the same configuration as base case
- DC: Unknown Parallel Operations.
- EP: Highly parallelizable, ↑ Configuration ↑ Gain
- SP: High Sync costs, ↓ Threads ↑ Gain

# CHOAMP: ENERGY



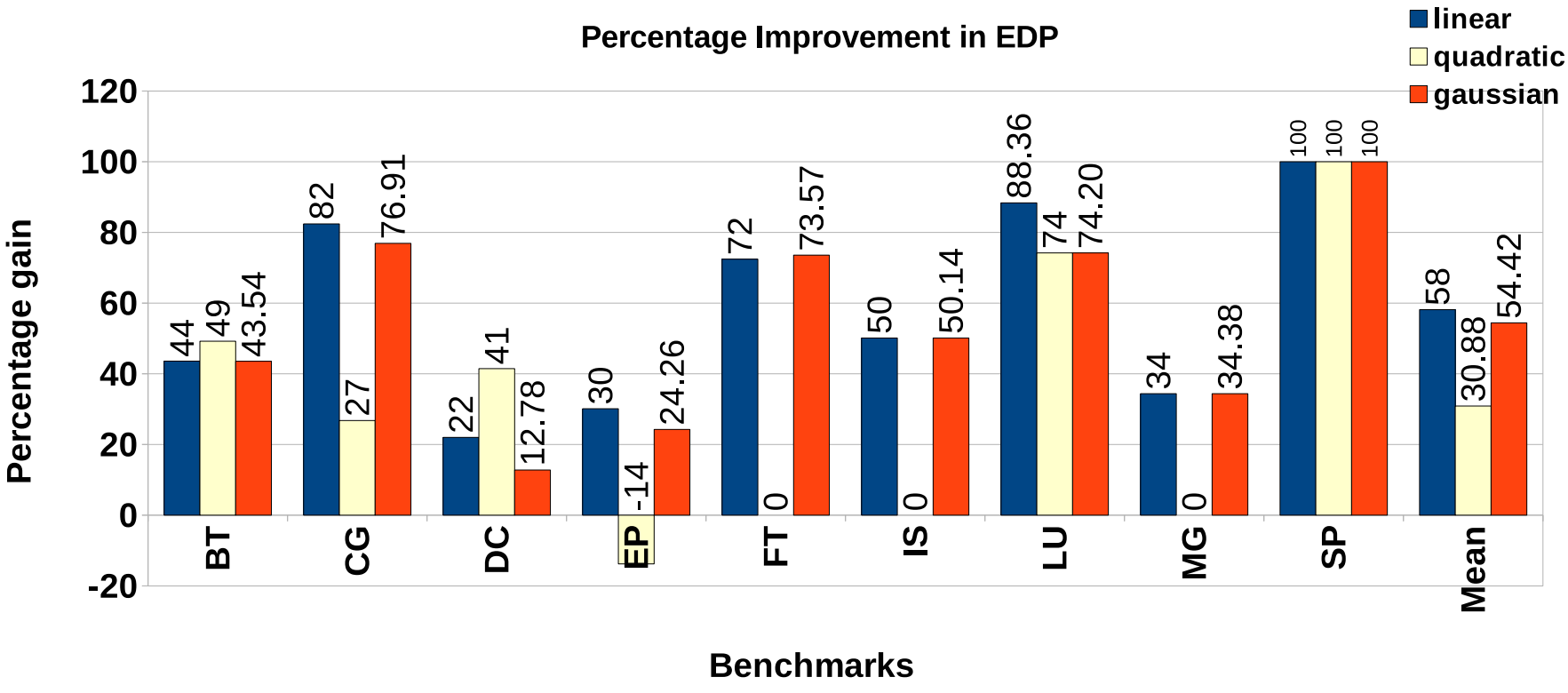

- On an average 65% improvement with Linear Regression `Predictor`
- Mostly Lower Configuration than base yields higher energy gains

# CHOAMP: ENERGY



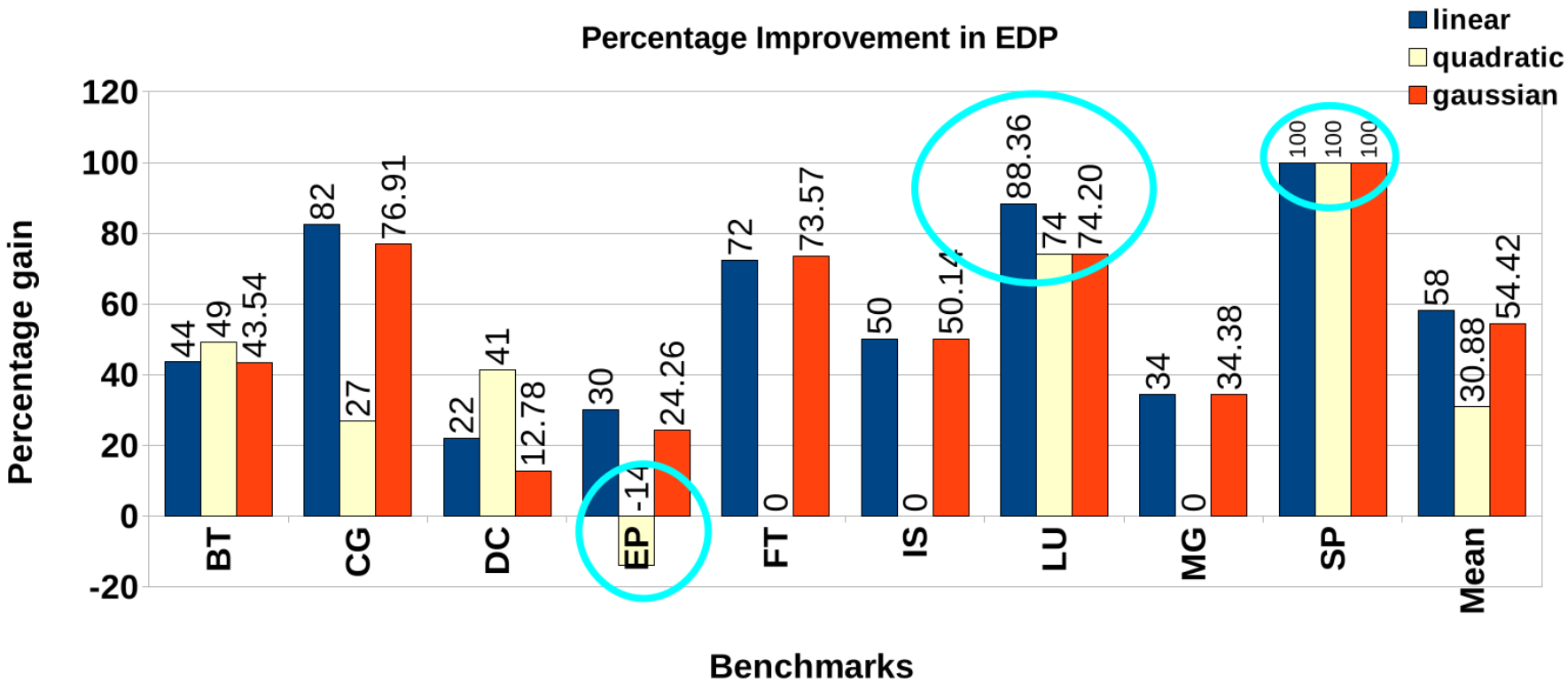

- EP: Highly parallel

# CHOAMP: EDP



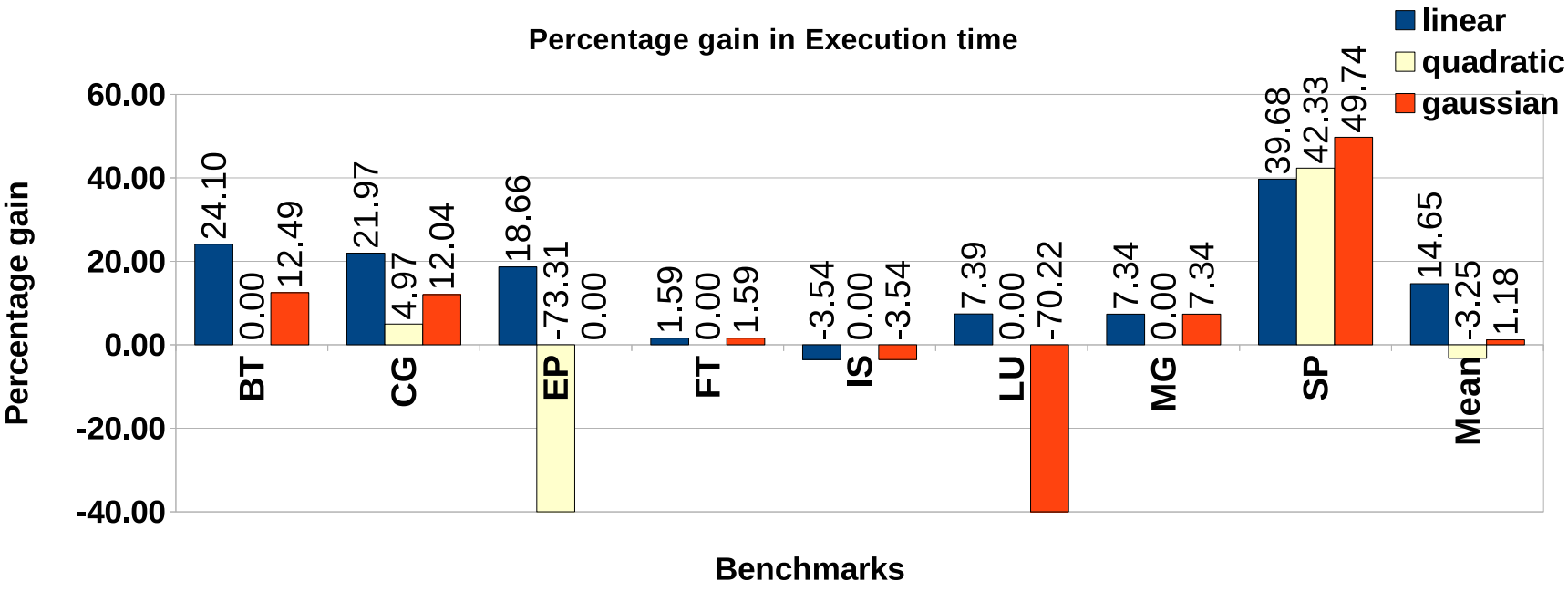

- On an average 54% improvement with Linear Regression Oracle
- Mostly a mid configuration yields good results

# CHOAMP: EDP



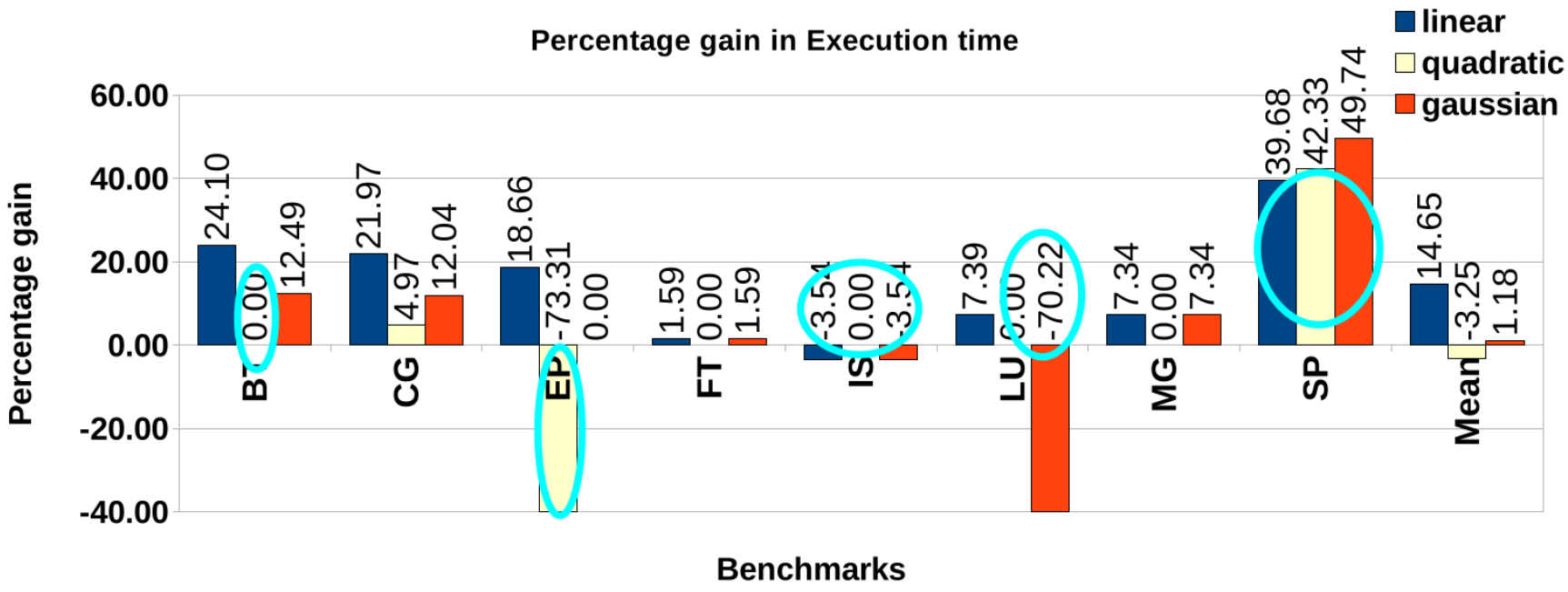

- LU: Gain higher than Energy, Execution Time negative
- SP: Rounded off

# CES + CHOAMP




- The trained Oracle without knowledge of CES
- On an average 14% improvement in execution time

# CES + CHOAMP




- BT: Base configuration selected
- EP, IS, LU: Loss Amplified, Better Load balancing
- SP: Gain reduced, Lower Sync costs

# CHOAMP:Conclusions

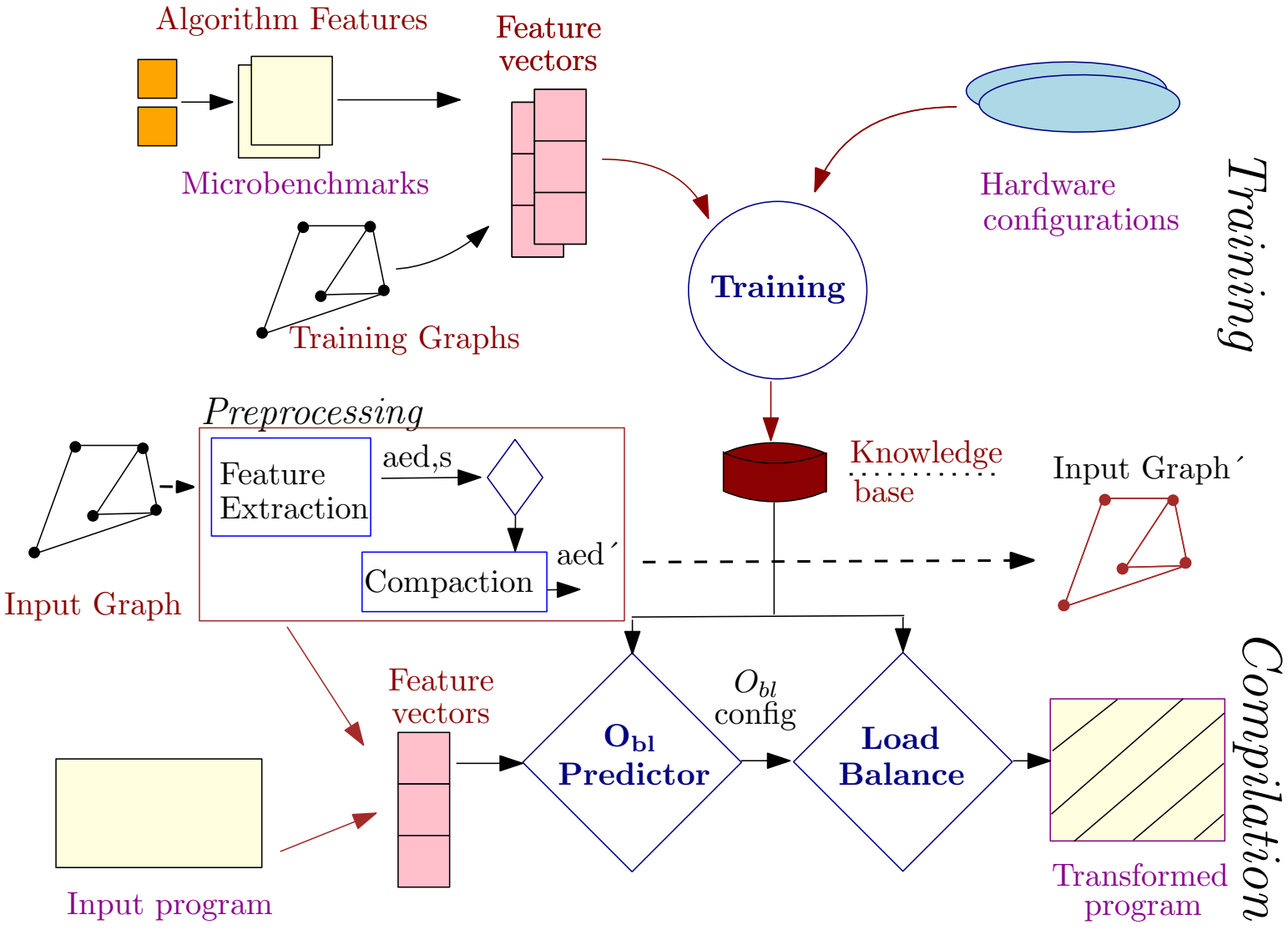
- On an average 28% improvement in execution time
- Average 65% improvement in energy consumption
- Average 58% improvement in EDP
- Average 14% improvement in execution time with CES
- Works well in tandem with dynamic scheduling algorithms (i) HMP (ii) CES.
- Sliding window regression: No significant gains.

# SIAM: Scheduling Irregular Workloads on Asymmetric Multicore processors

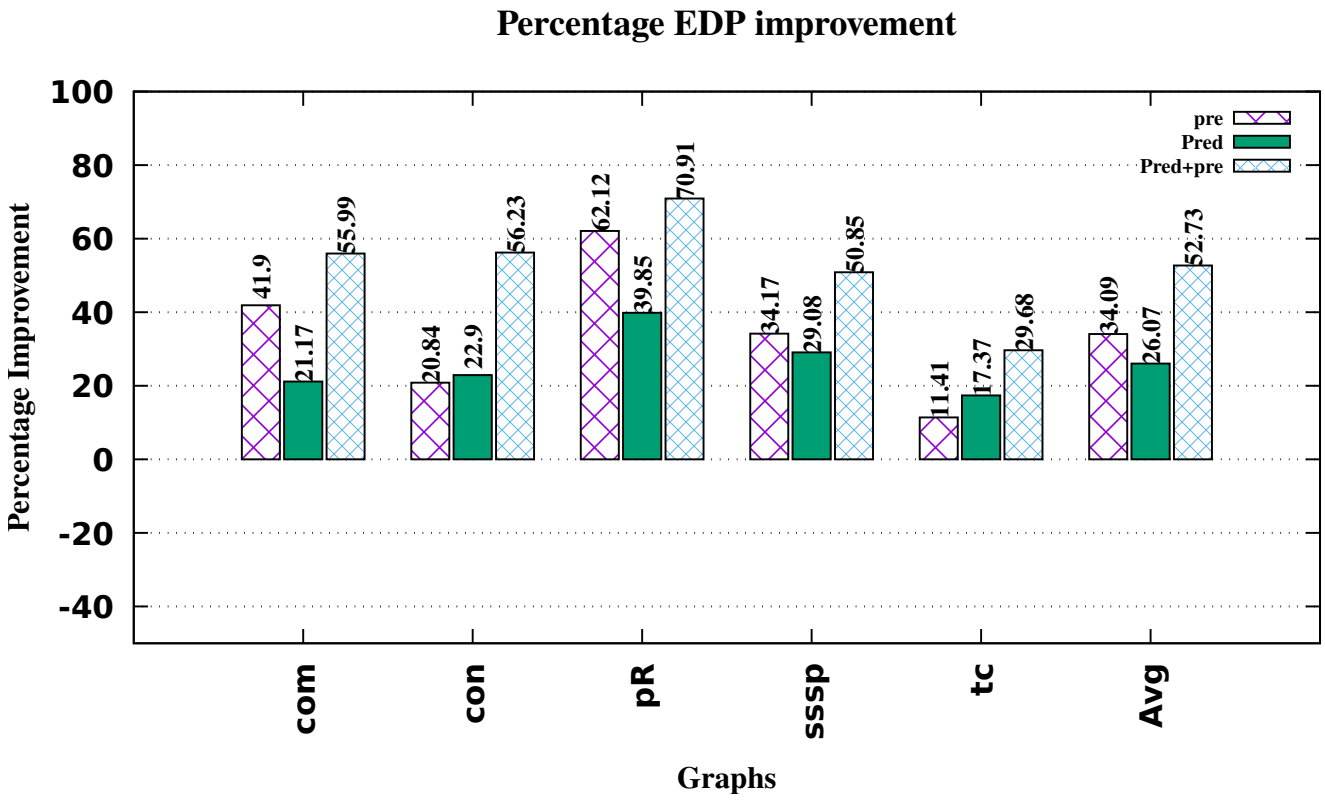
# Irregular Parallel Workloads: Graph Algorithms

- Parallelism depends on the input graphs.
- Include Graph into training the oracle.
- How to represent the graph with set of properties?
- Should capture:
  - Overall Workload
  - Workload distribution
  - Memory accesses/Atomics
- Training for Algorithm-Graph pair

# SIAM: Workflow



# Average EDP gain over Algorithms




Higher the better.

# SIAM: Conclusion

- Graph properties are required to fully capture the workload.
- Compaction: Good representation of the input graph can reduce both energy consumption and execution time.
- Predictor shows on an average 52% improvement in EDP consumption.
- Add edge weight Properties : for weight based algorithms. SSSP, MST, Max flow etc