Greedy Talents Method of Multi-Core CPU Usage

- iWAPT 2018 –

25 May 2018

Tim Platt Zhiliu Yang Chen Liu

GLOBALFOUNDRIES®

Clarkson

CAMEL: Computer Architecture and Microprocessor Engineering Lab

What Are We Trying to Solve

- Given N programs (to be run lots of times)…
  - How many processors to assign each program to achieve:
    - Best performance
    - Lowest energy / EDP
  - What voltage / frequency setting to assign each processor to achieve:
    - Best performance
    - Lowest energy / EDP
  - No prior program knowledge

Outline

- Test Platform Description
- Algorithm Description
- Test Validation
- Summary of Program Runs
- Results from 4 Programs (mpi_mm, conv_time6, cpi, seqpi2)
- Comparison to Previous Results
- Conclusions / Contribution

Test Platform & Rational

- Test platform was Intel Single-Chip Cloud Computer (SCC)
  - Older experimental processor but
  - Similar to previous work
    - Good for comparisons

    - Future work could repeat on new platform
      - Xeon?

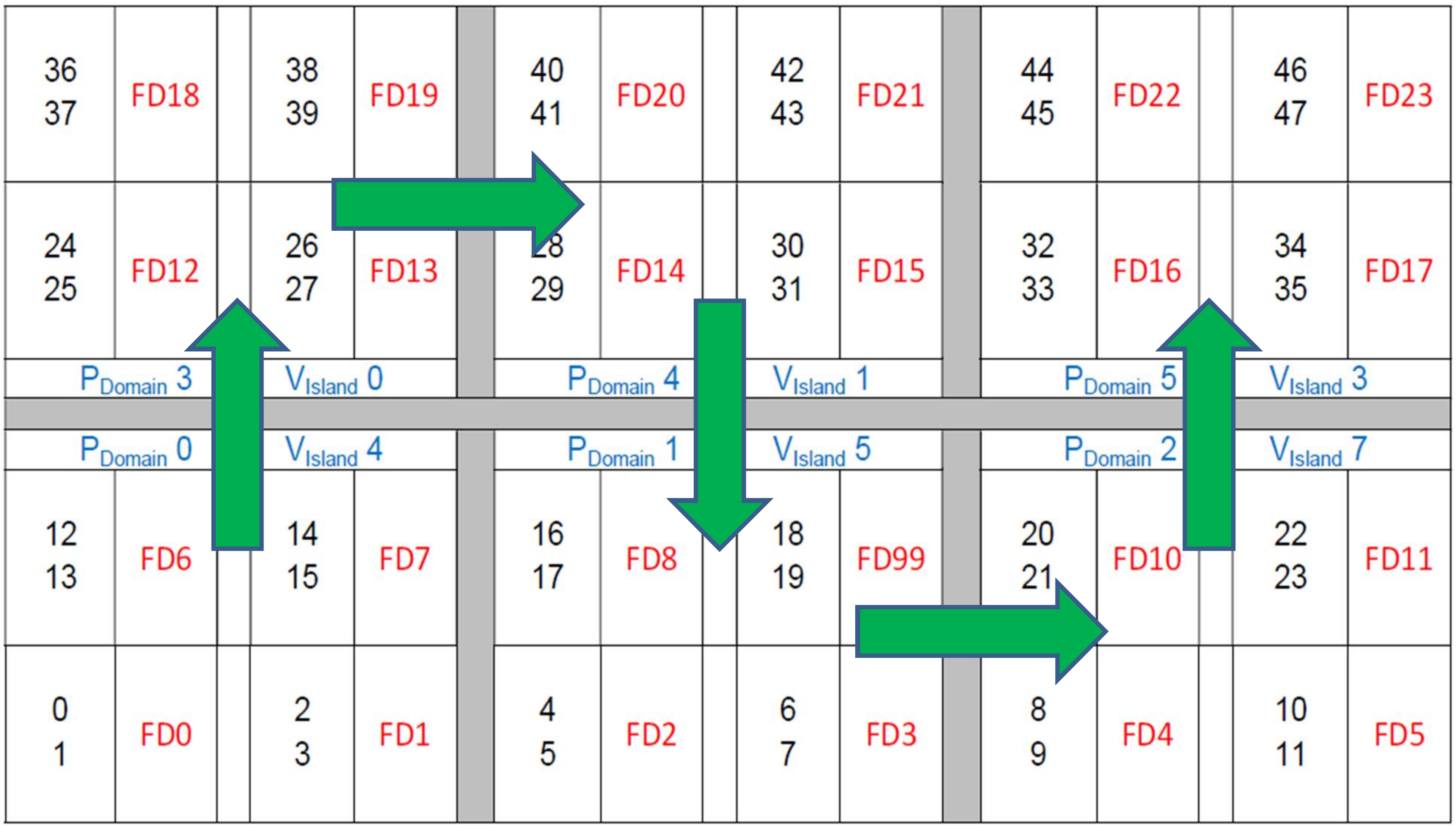
Test Platform

- Intel SCC
  - 48 Pentium (P54C) class processors
    - 32 KB L1 cache
    - 256 KB L2 cache
  - Message passing for communication
  - Each pair of processors shares frequency setting
  - 6 voltage islands
  - 5 freq./voltage setting (gears)

Gear Settings

Gear	Freq. (MHz)	Voltage (V)
1	800	1.1
2	533	1.0
3	400	0.9
4	320	0.8
5	267	0.8

SCC Domains



36 37
FD18
38 39
FD19
40 41
FD20
42 43
FD21
44 45
FD22
46 47
FD23
24 25
FD12
26 27
FD13
29
FD14
30 31
FD15
32 33
FD16
34 35
FD17
P_{Domain} 3
V_{Island} 0
P_{Domain} 4
V_{Island} 1
P_{Domain} 5
V_{Island} 3
P_{Domain} 0
V_{Island} 4
P_{Domain} 1
V_{Island} 5
P_{Domain} 2
V_{Island} 7
12 13
FD6
14 15
FD7
16 17
FD8
18 19
FD99
20 21
FD10
22 23
FD11
0 1
FD0
2 3
FD1
4 5
FD2
6 7
FD3
8 9
FD4
10 11
FD5

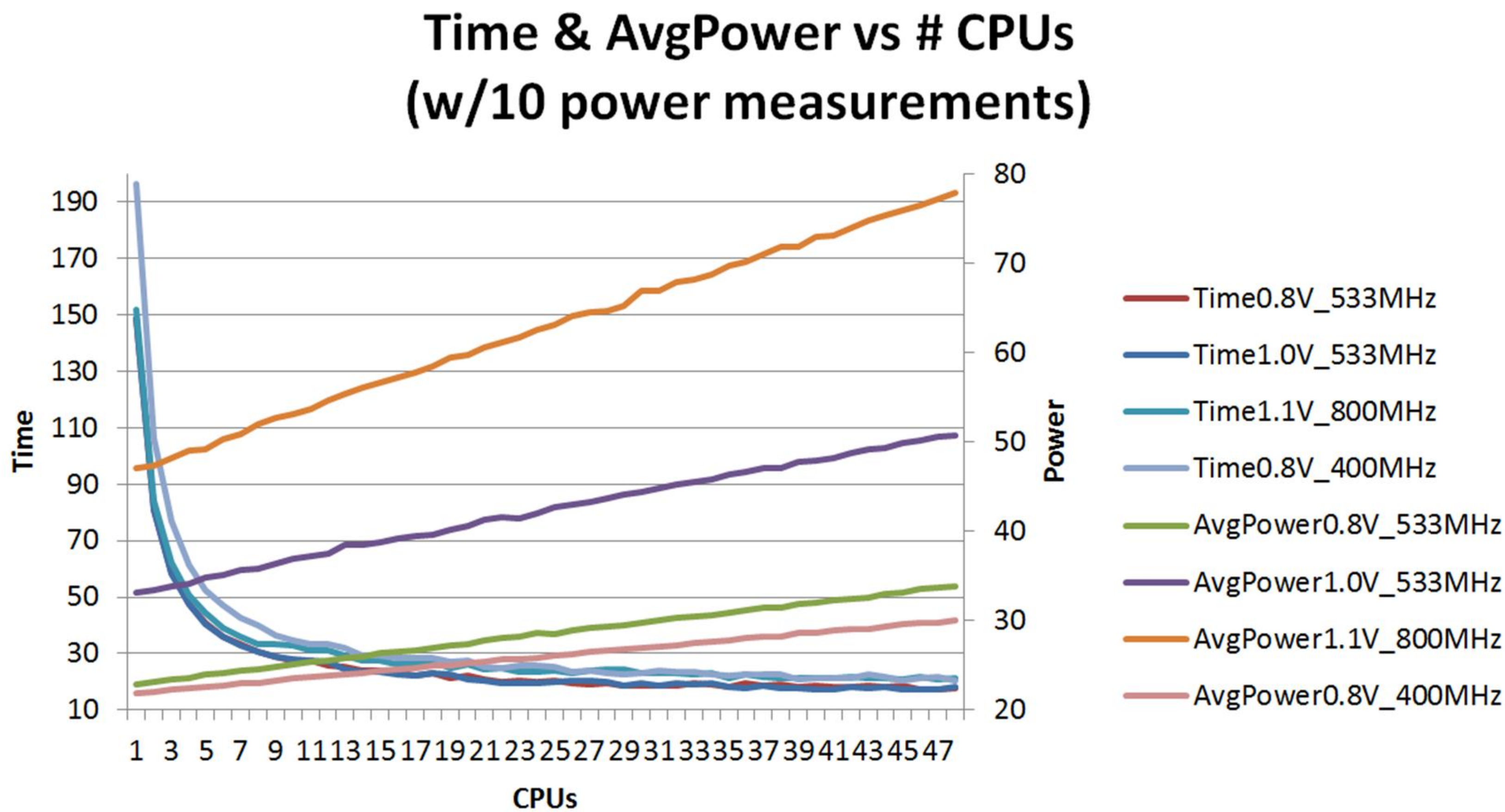
SCC “Host” Computer

- MCPC
  - GreedyTalents runs on MCPC
    - Evaluation programs run on SCC
  - Enables control of frequency of SCC processors
  - Enables control of power of SCC processors
    - Provides power measurement of complete SCC

Evaulation Programs

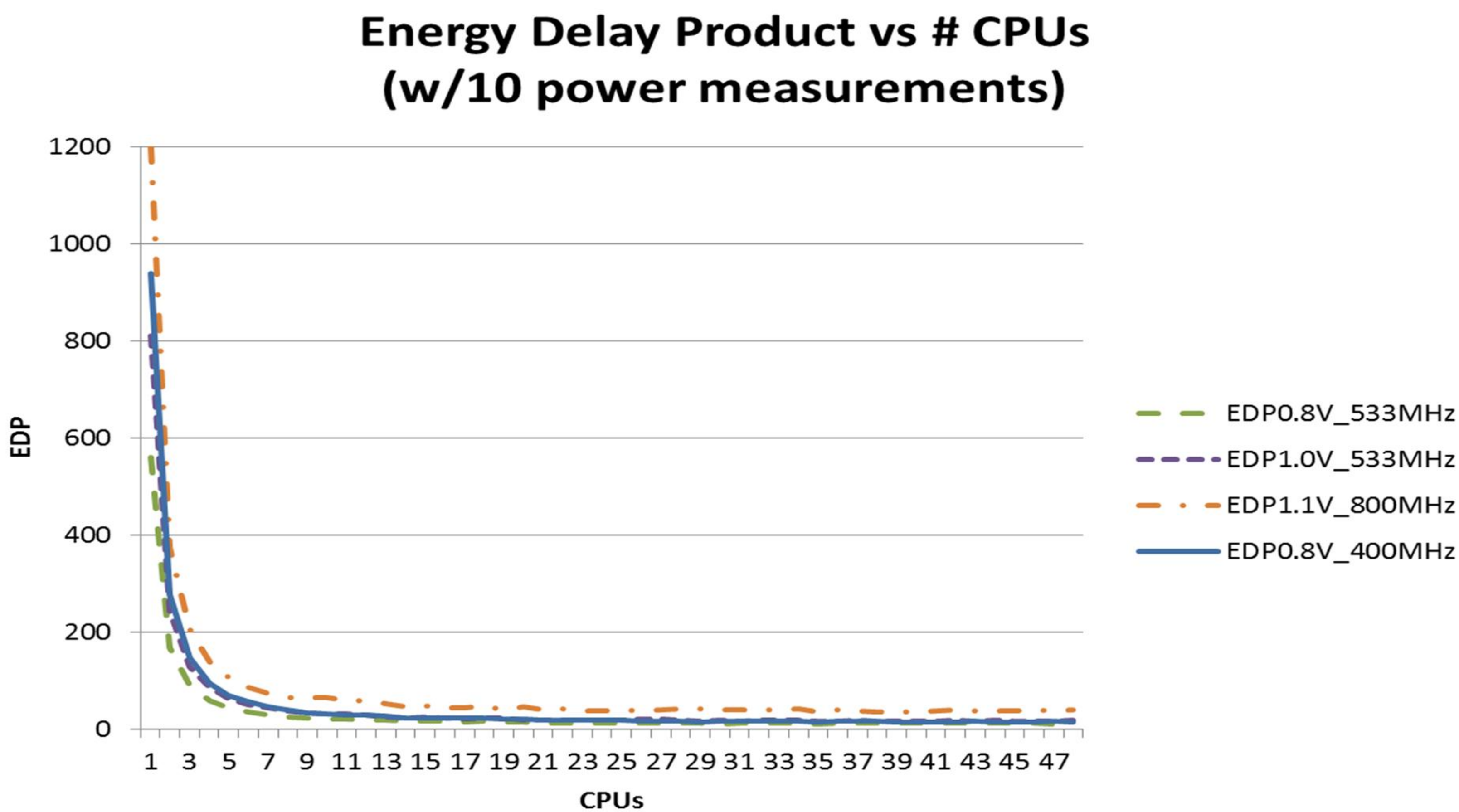
- Four programs (same as previous work)
  - mpi_mm
    - matrix multiplication
  - conv_time6 256
    - convolution computation
  - cpi 1000000000
    - compute π (highly parallel)
  - seqpi2
    - compute π (highly serial)

Observations



- Number of CPUs provides largest performance improvement

Observations



- Freq./Voltage provides minimal EDP change

Conclusions from Observations

- Focus on the number of CPUs
  - Even small amount of additional CPUs provides dramatic performance increase
    - and corresponding Energy / EDP reduction
      - EDP = f(time)
  - More CPUs provides lots of performance increase
    - In the steep part
      - Let the number of CPUs be greedy
- Secondary effect for Freq./Voltage
  - "Fine tuning" of EDP

GreedyTalents Algorithm (pseudo code)

- Assume workload is < available processors
  - e.g. 6-8 programs vs 48 available (more in future)
  - Start with fastest performance settings (F&V)

1) Each program gets 1 CPU (or minimum needed)
   - Run and evaluate performance
2) Give each program additional CPUs (1 to start)
   - Run and evaluate Δperformance
3) Evaluate derivative of performance change
   - Steepness of derivative

GreedyTalents Algorithm (cont'd)

4) Assign new number of CPUs based on performance derivative
   - Large improvement = lots of additional CPUs
     - Maximum amount of GPUs = greediness
   - Medium improvement = medium additional CPUs
   - Small improvement = 1 or 0 additional CPUs
   - Performance increase – return to previous number of CPUs
5) Iterate 2-4) until no program is given more CPUs or no more CPUs

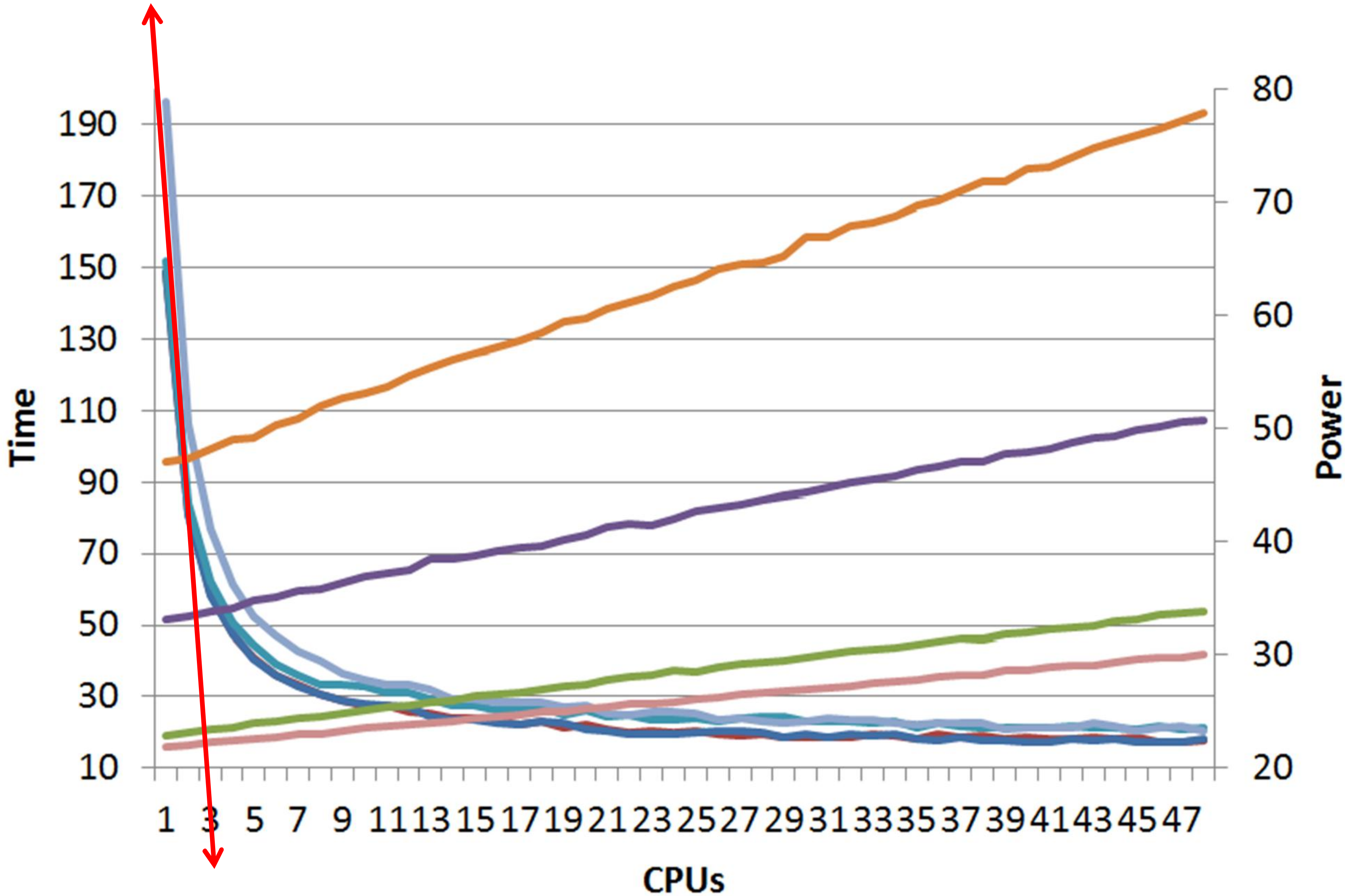
GreedyTalents Algorithm (cont'd)

7) Adjust frequency (& voltage) such that:
   – Lots of CPUs set for lower frequency
   – Fewer CPUs set for higher frequency
   – Slowest program gets no adjustment

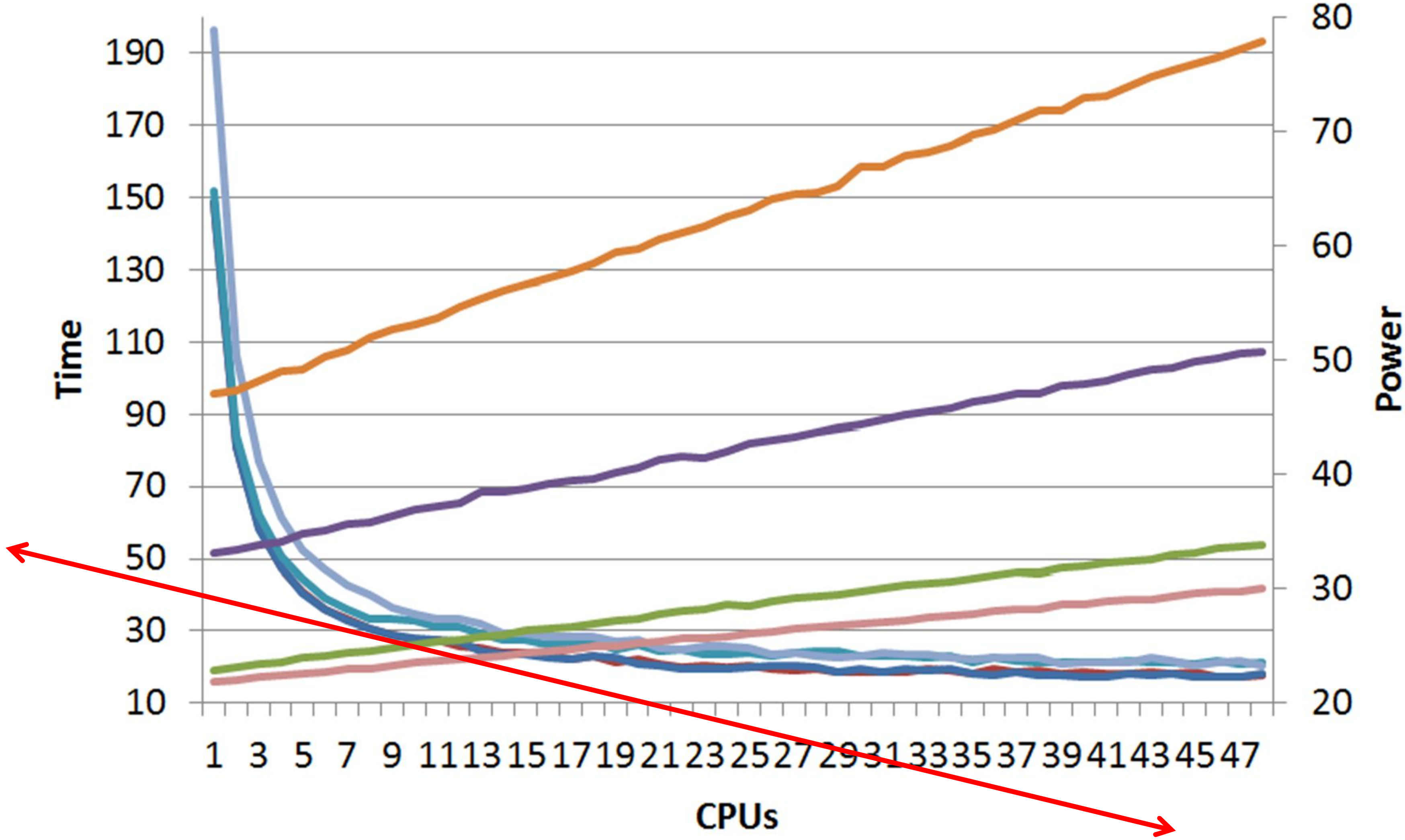
- When assigning additional CPUs
  – Max increase – algorithm input (greediness)

Example Algorithm Application

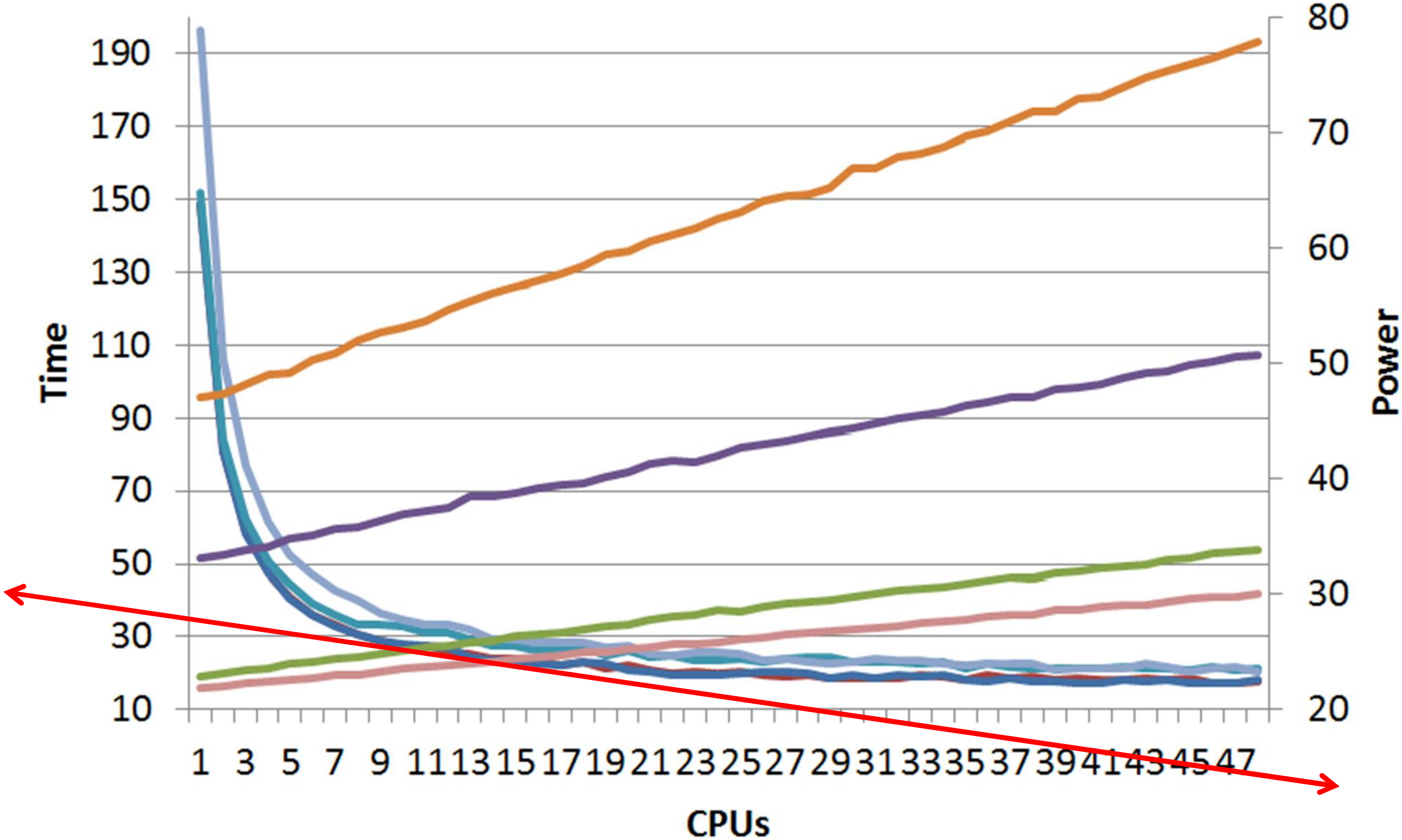
Phase I – Iteration 1



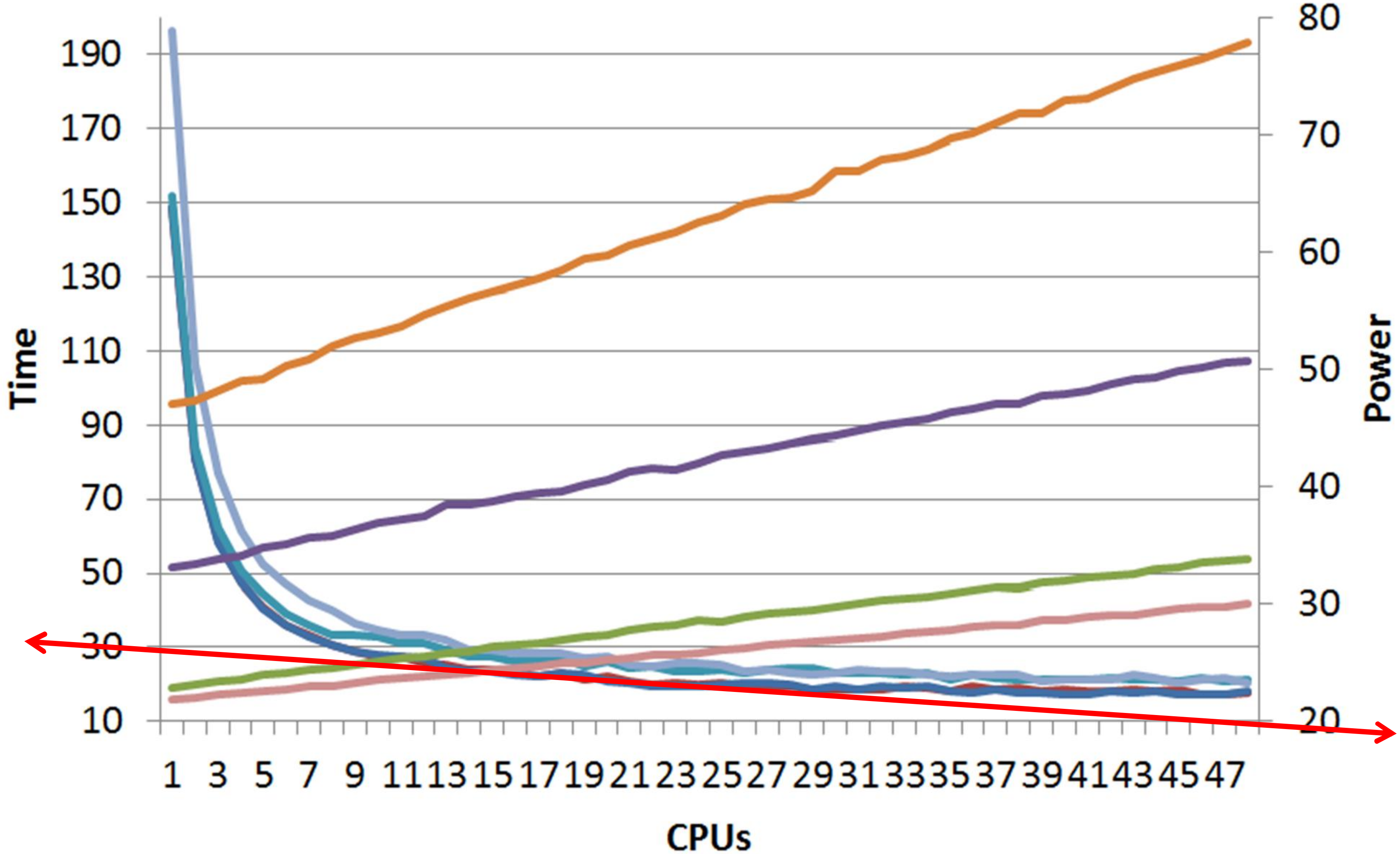
Phase I – Iteration 2



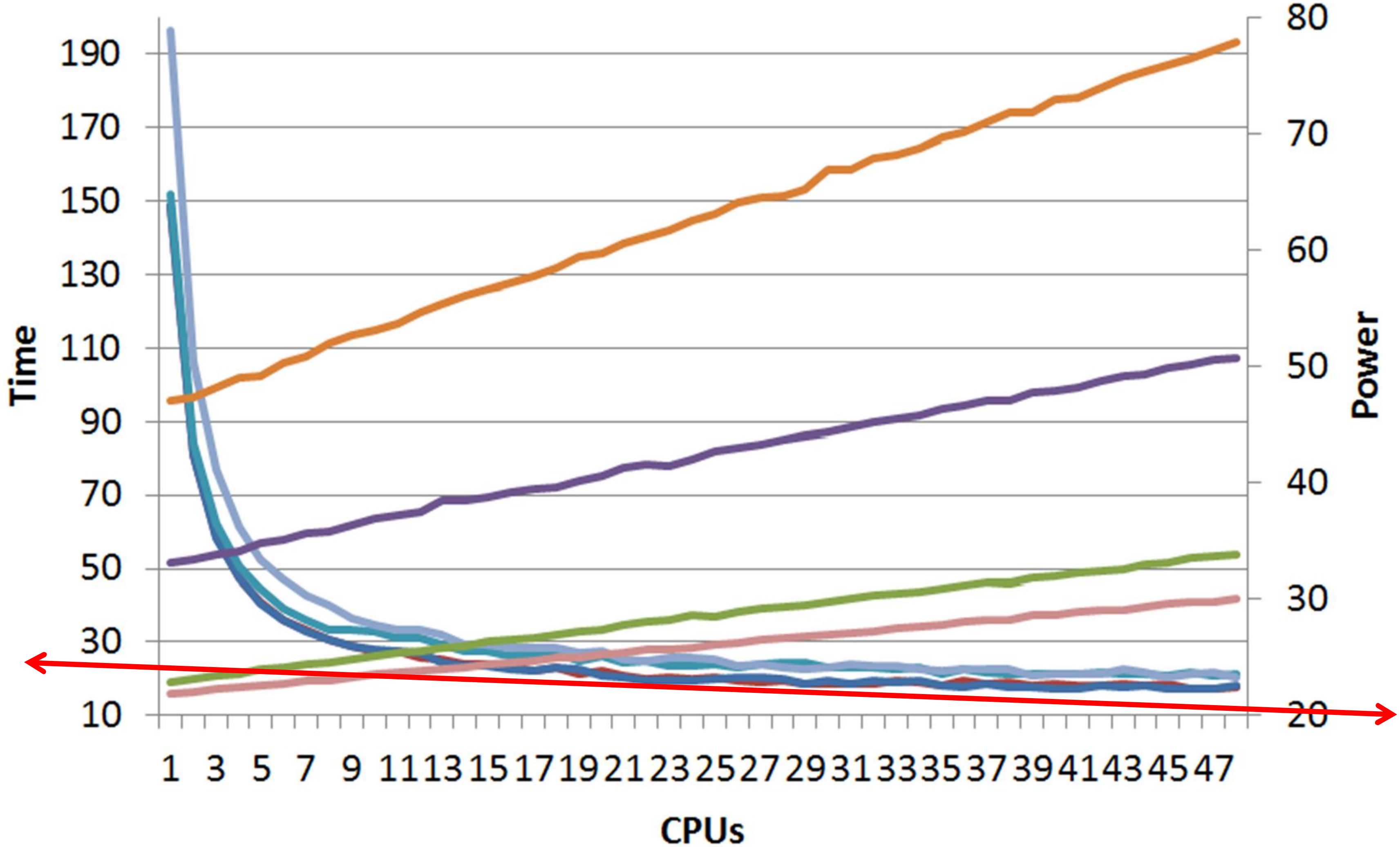
Phase I – Iteration 3



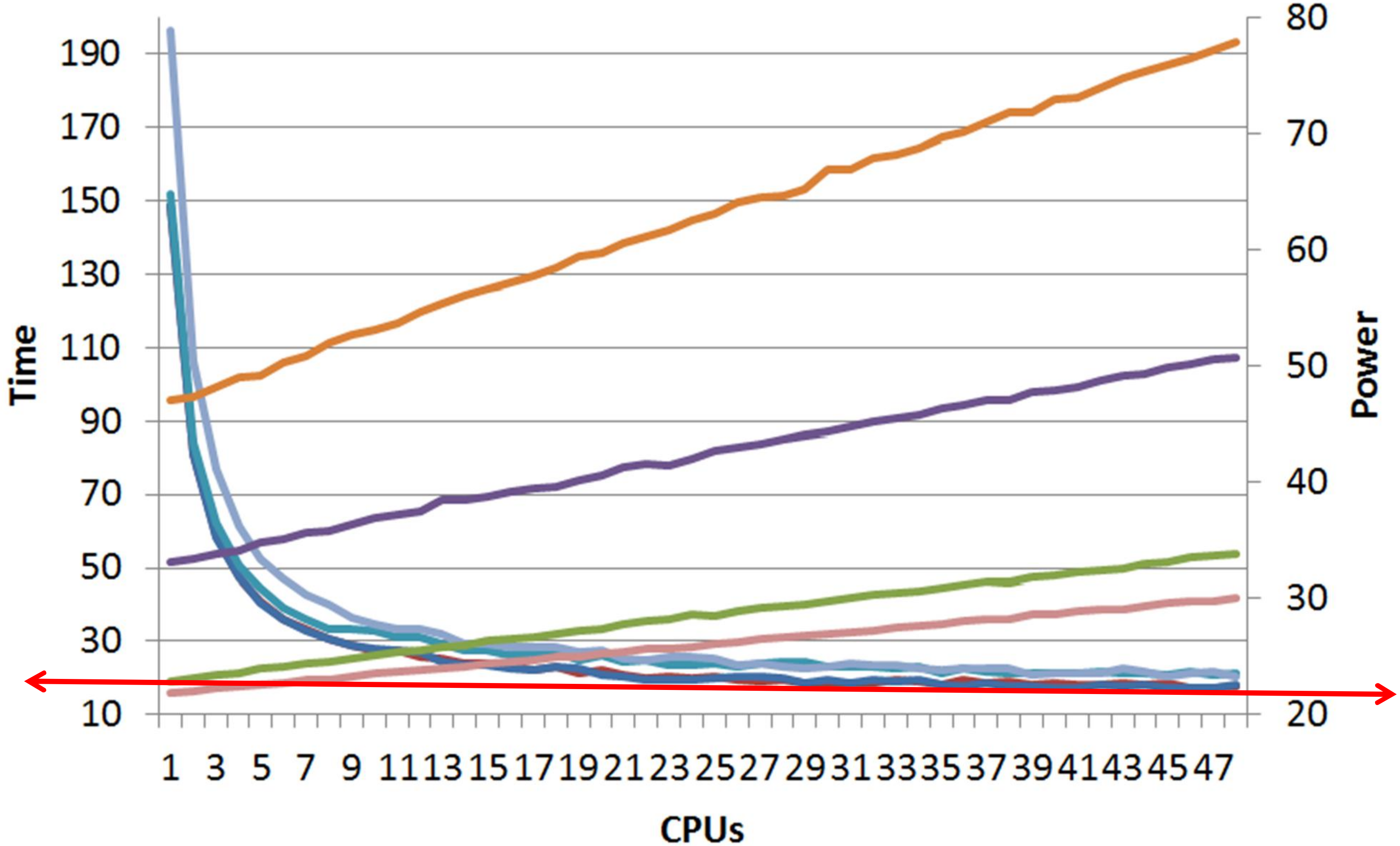
Phase I – Iteration 4



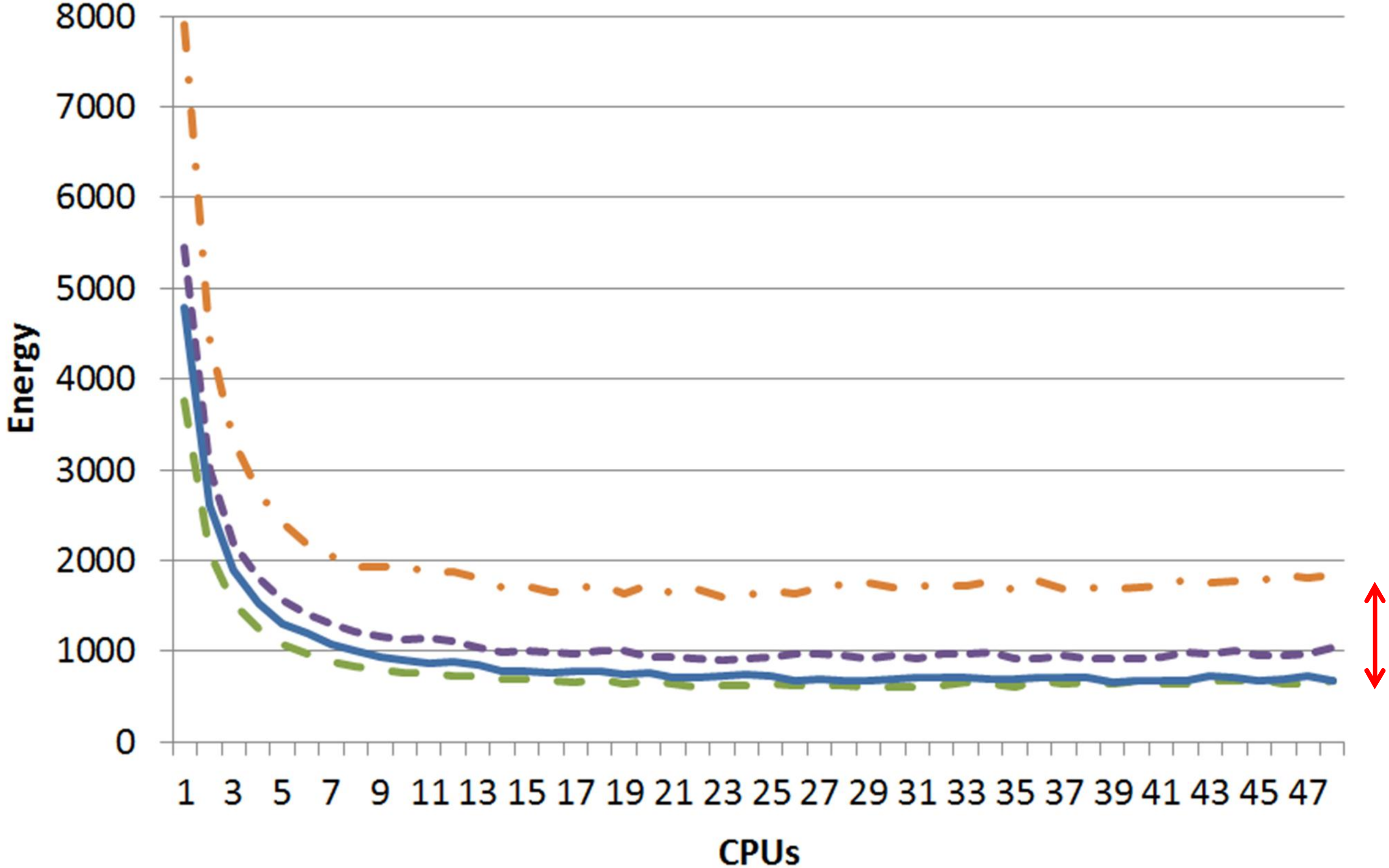
Phase I – Iteration 5



Phase I – Iteration 6



Phase II



8000
7000
6000
5000
4000
3000
2000
1000
0
Energy
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47
CPUs

“Secret Sauce”

- $Exec\_Time_{Goal} = Exec\_Time_{Prev} \cdot Ambition$

- $\#CPUs = \frac{Exec\_Time - Exec\_time_{Prev}}{Exec\_Time_{Goal}} \cdot Greediness$

Test Program Description

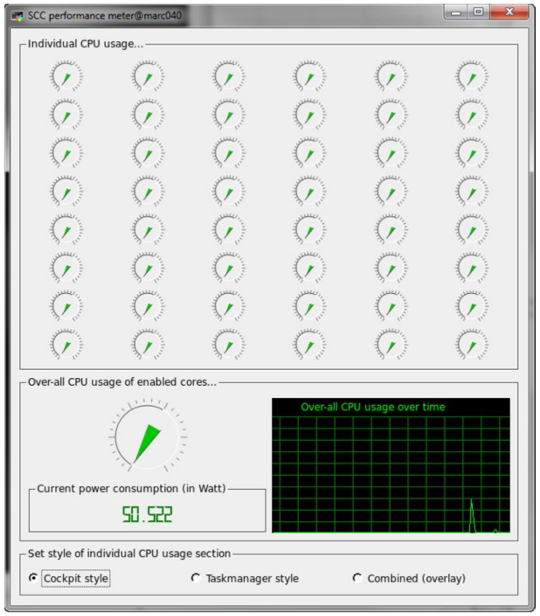
- SCC programs to run provided by file
  - VERY easy to change
  - Also specifies starting # of CPUs and priority
- User specifies greediness (i.e. max # CPUs that can be added in a single generation)
  - Allows for experimentation
- User specifies time between power measurements
- User specifies maximum # of generations
- Lots of other features to support experimentation, debug

Test Program Description (cont'd)

- GreedyTalents is written in C
- GreedyTalents runs on MCPC
- Uses "fork" to start programs on SCC
- Time measured by program on MCPC
  - from 1st program launch
- Power measured by program on MCPC
  - Continuously once all programs are launched
    - Time between power measurements is user specified

Test Validation

- Ran 3 experiments with known execution time
  - **sleep 15** (all four programs)
- Manually measured power w/sccPerf
- Checked results
  - Execution time
  - Power
  - Energy
  - EDP



Test Validation Summary

- Avg Pwr: ≈ 50.6 – 51.0 W
- Total Time: ≈ 16.25 s
  - Pgm1: ≈ 16.25 s
  - Pgm2: ≈ 16.25 s
  - Pgm3: ≈ 16.25 s
  - Pgm4: ≈ 16.25 s

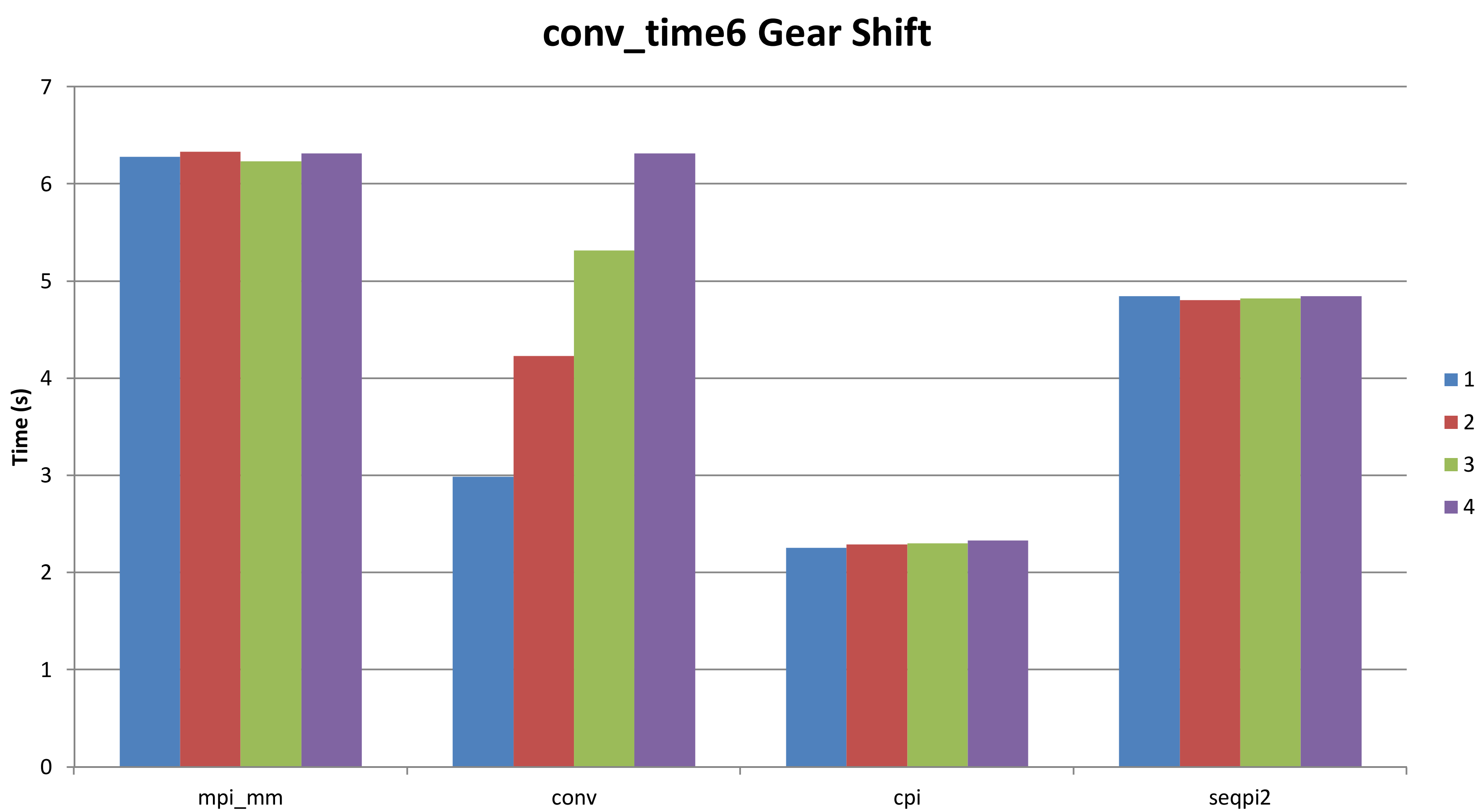
≈ 1.25 s

- overhead

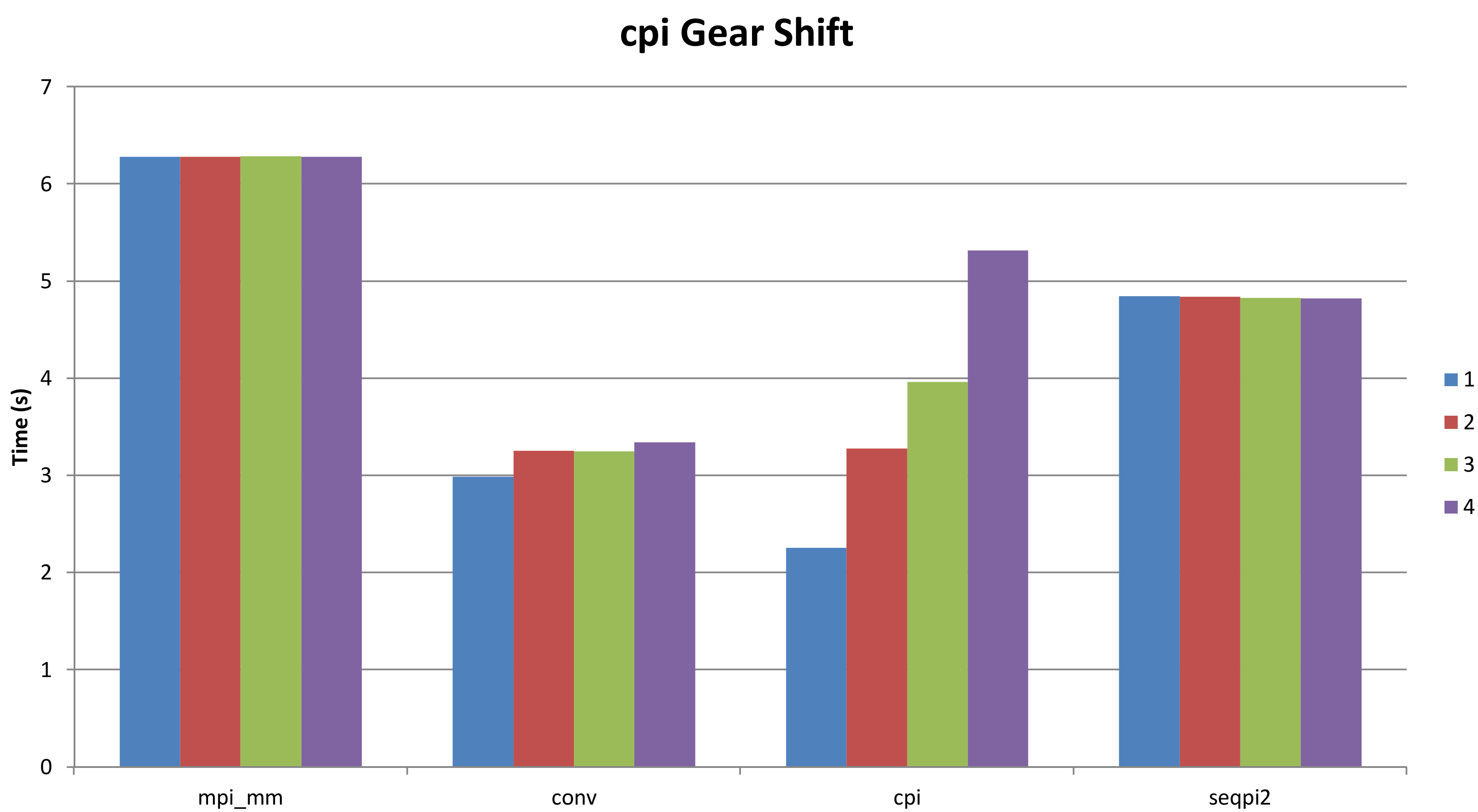
Phase II Validation

- Employed “-user_gears” to set only one program to a different (i.e. lower gear) and observed the results
  - Expected and observed a slower run time
  - Tried gear 1, 2, 3, 4
  - Confirmed for all four programs

Phase II Validation – Pgm2

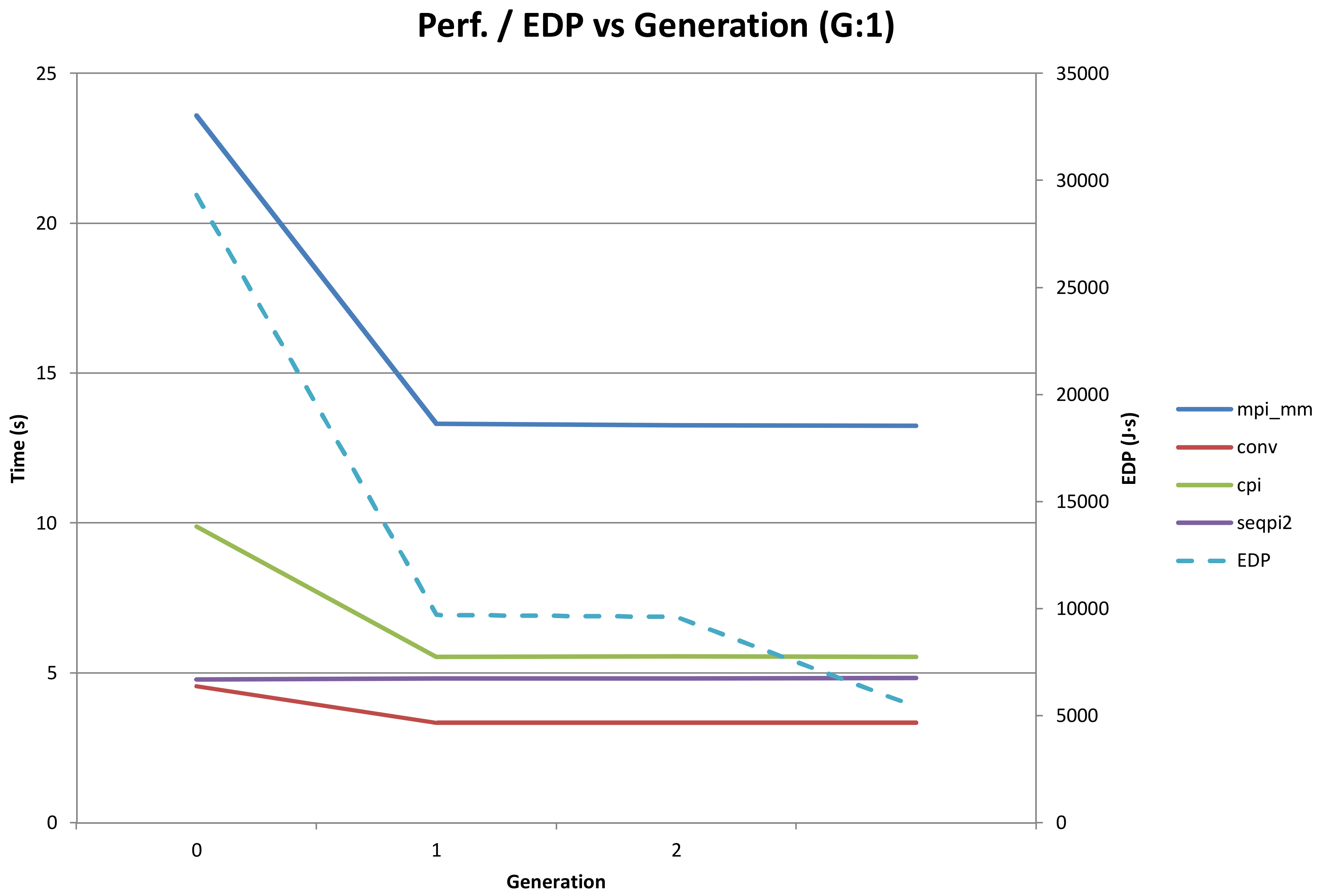


Phase II Validation – Pgm3

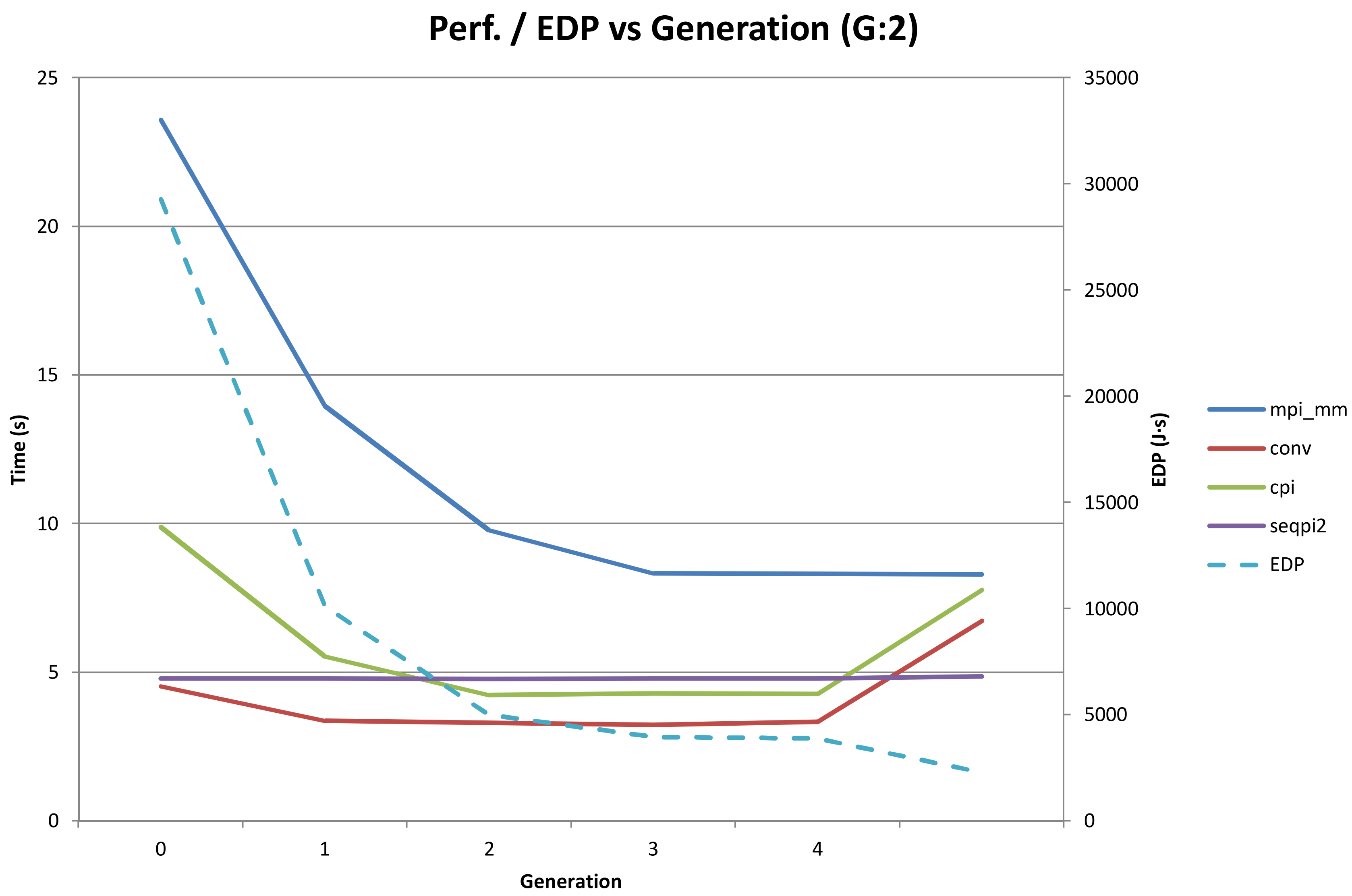


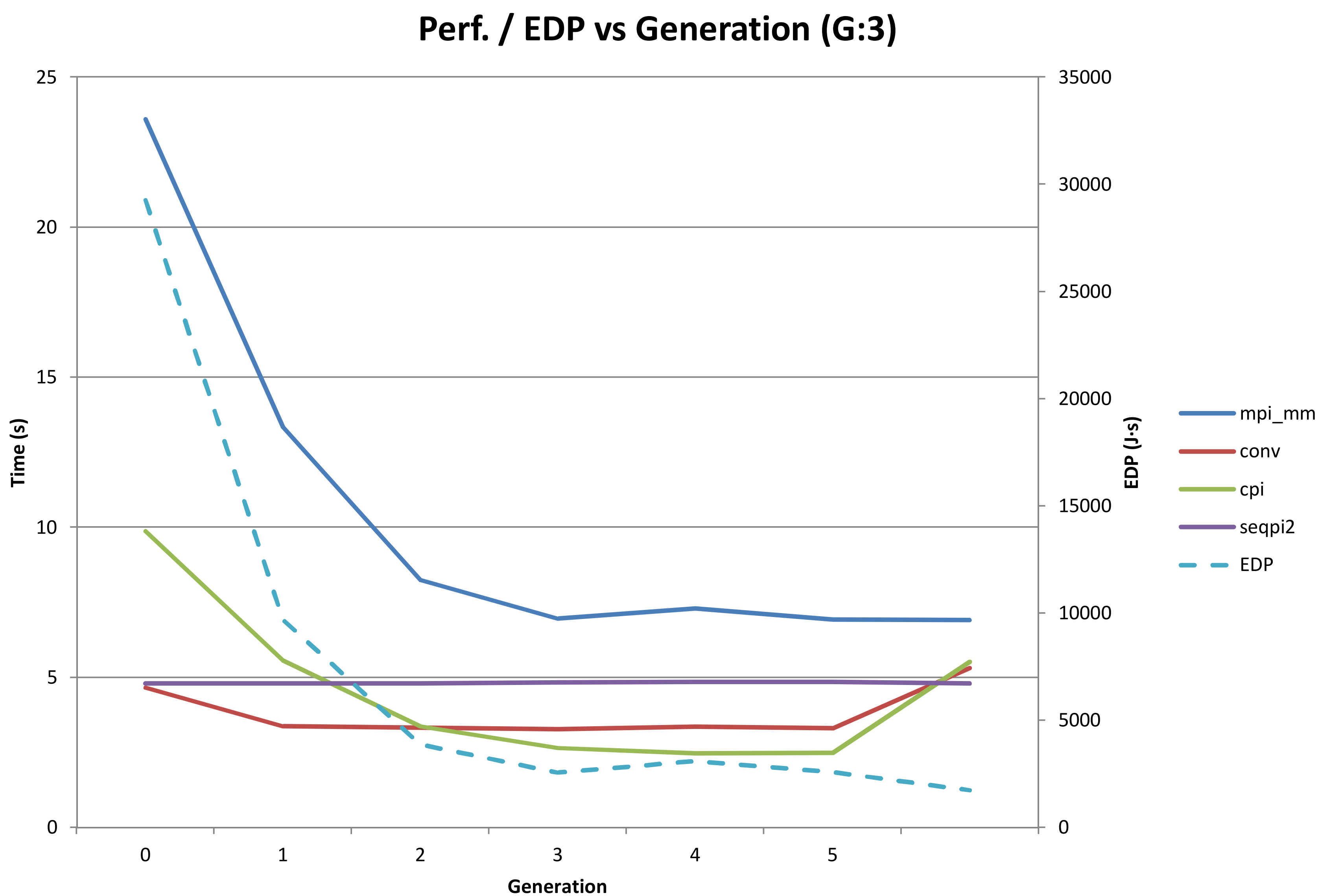
Experiments

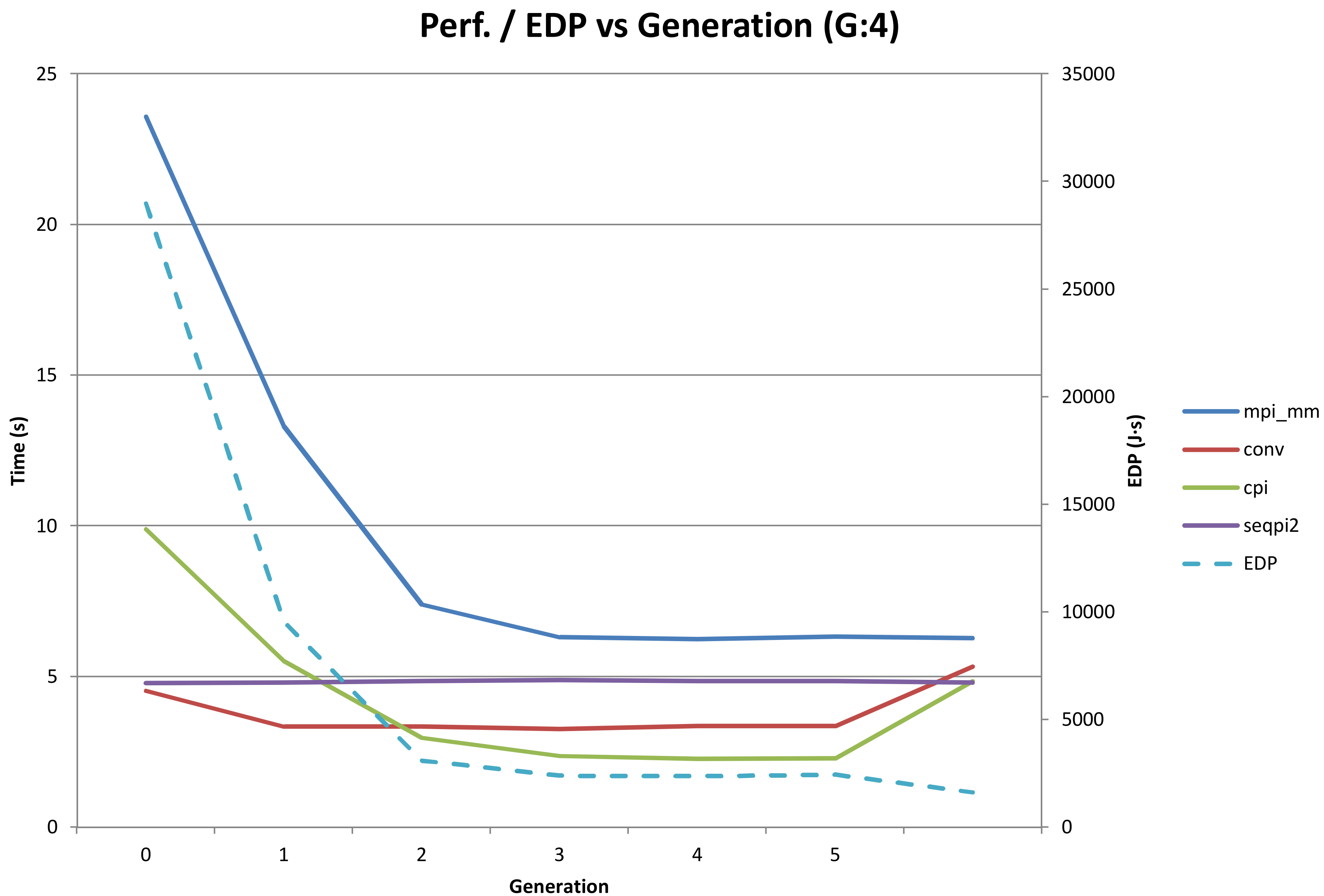
- Ran GreedyTalents with increasing amounts of greediness



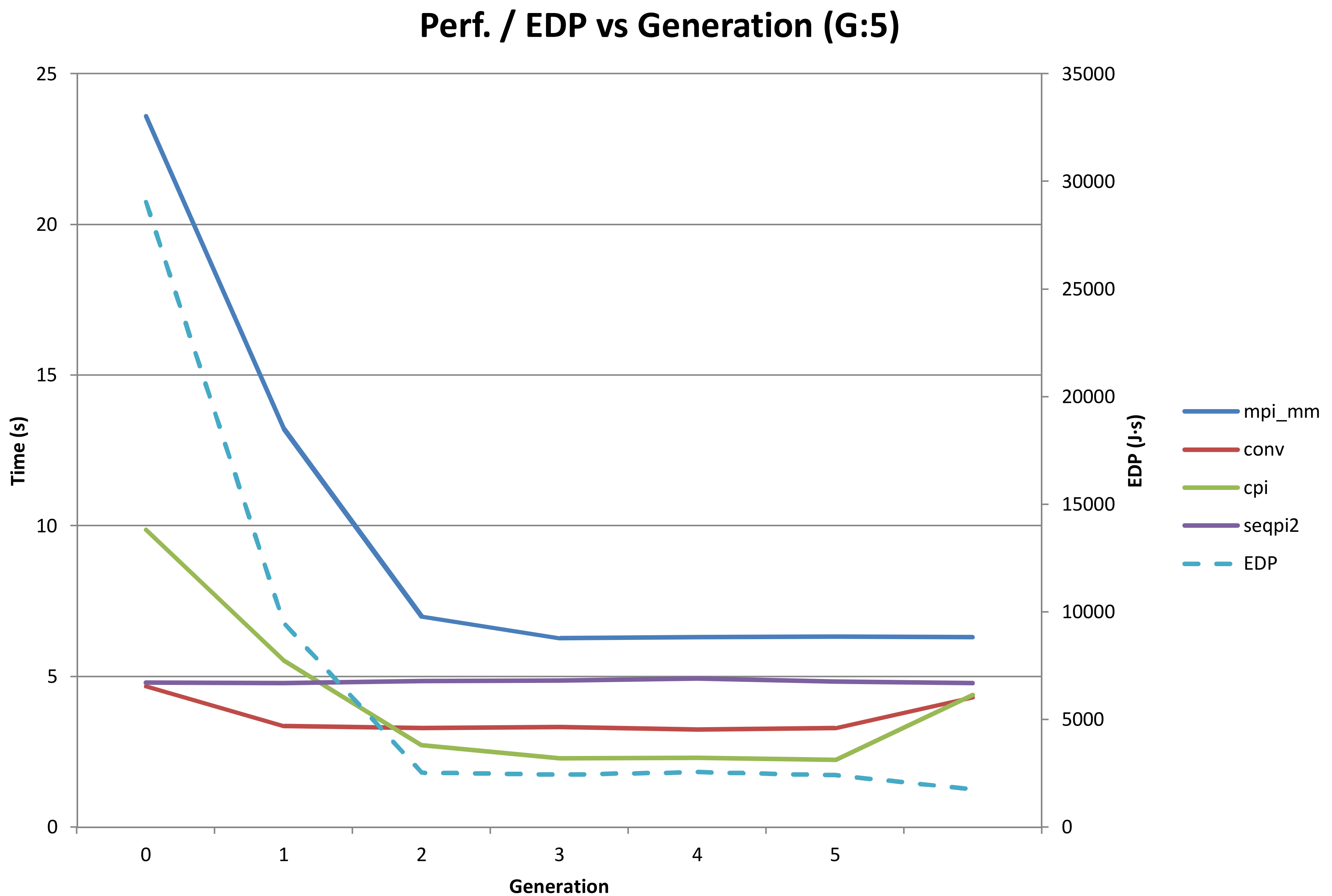
Perf. / EDP vs Generation (G:1)
Time (s)
25
20
15
10
5
0
EDP (J·s)
35000
30000
25000
20000
15000
10000
5000
0
0
1
2
Generation
mpi_mm
conv
cpi
seqpi2
EDP

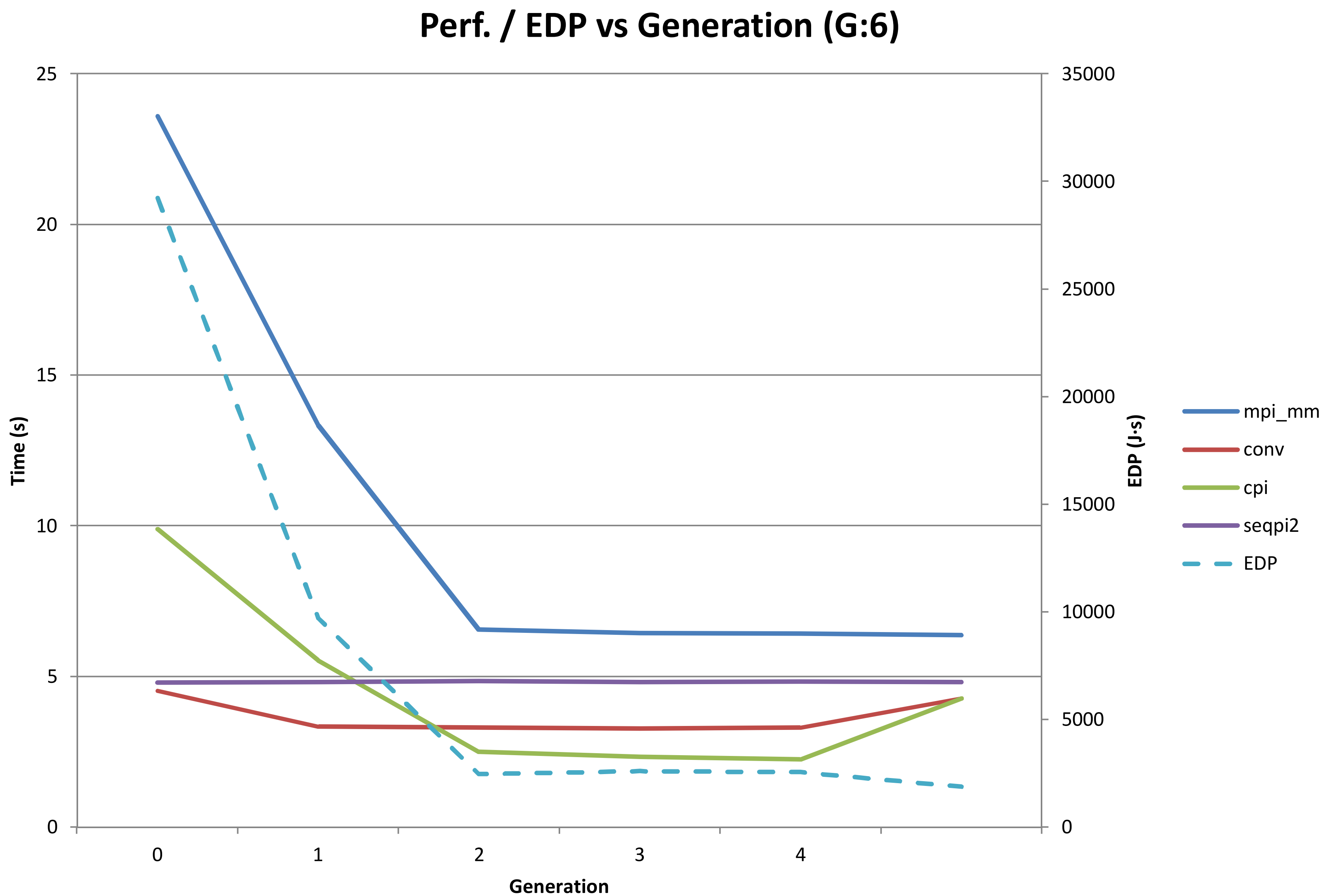

Perf. / EDP vs Generation (G:2)
Time (s)
EDP (J·s)
Generation
0
5
10
15
20
25
0
5000
10000
15000
20000
25000
30000
35000
0
1
2
3
4
mpi_mm
conv
cpi
seqpi2
EDP


Perf. / EDP vs Generation (G:3)
Time (s)
EDP (J·s)
Generation
0
5
10
15
20
25
0
5000
10000
15000
20000
25000
30000
35000
0
1
2
3
4
5
mpi_mm
conv
cpi
seqpi2
EDP

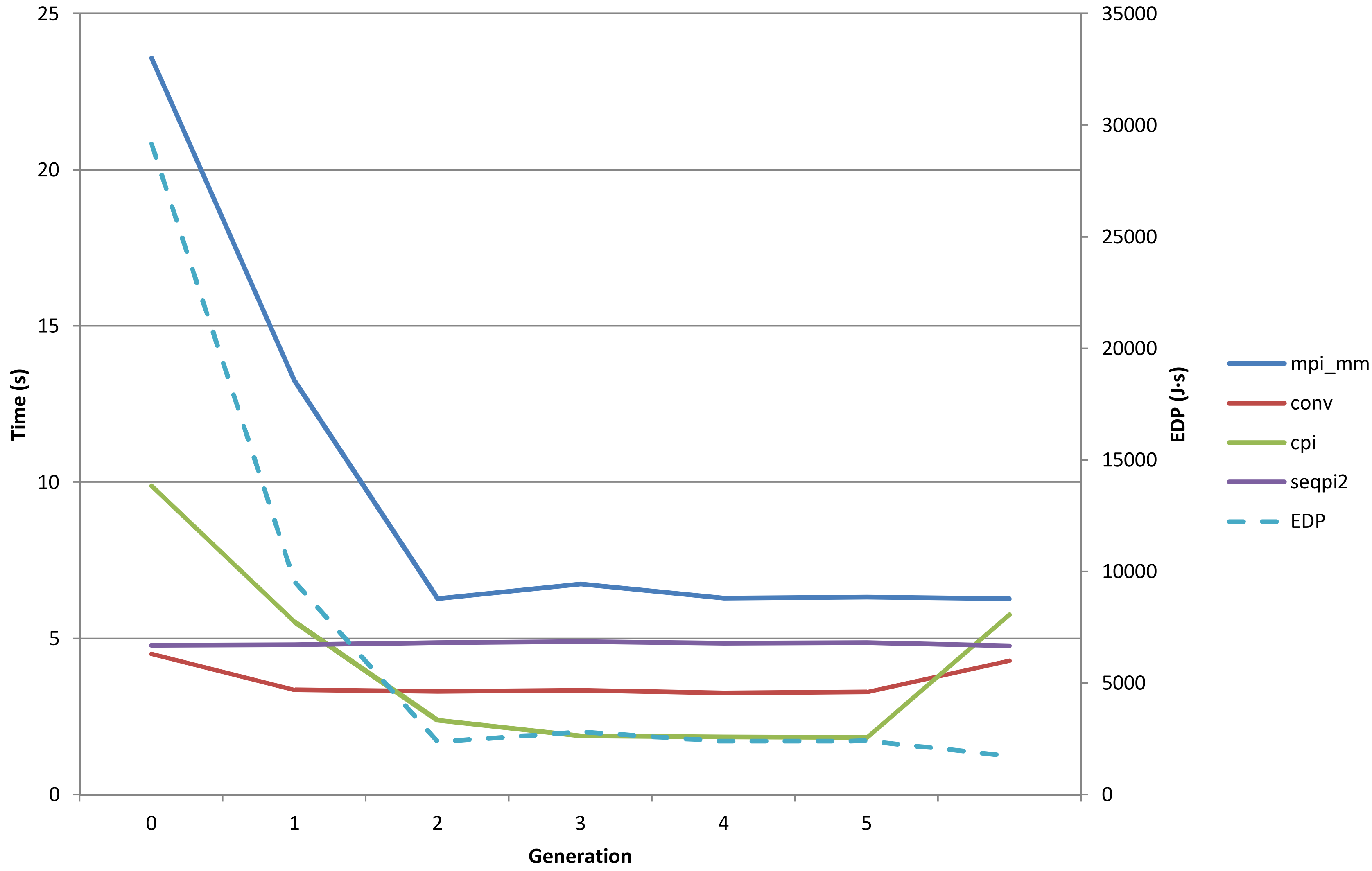


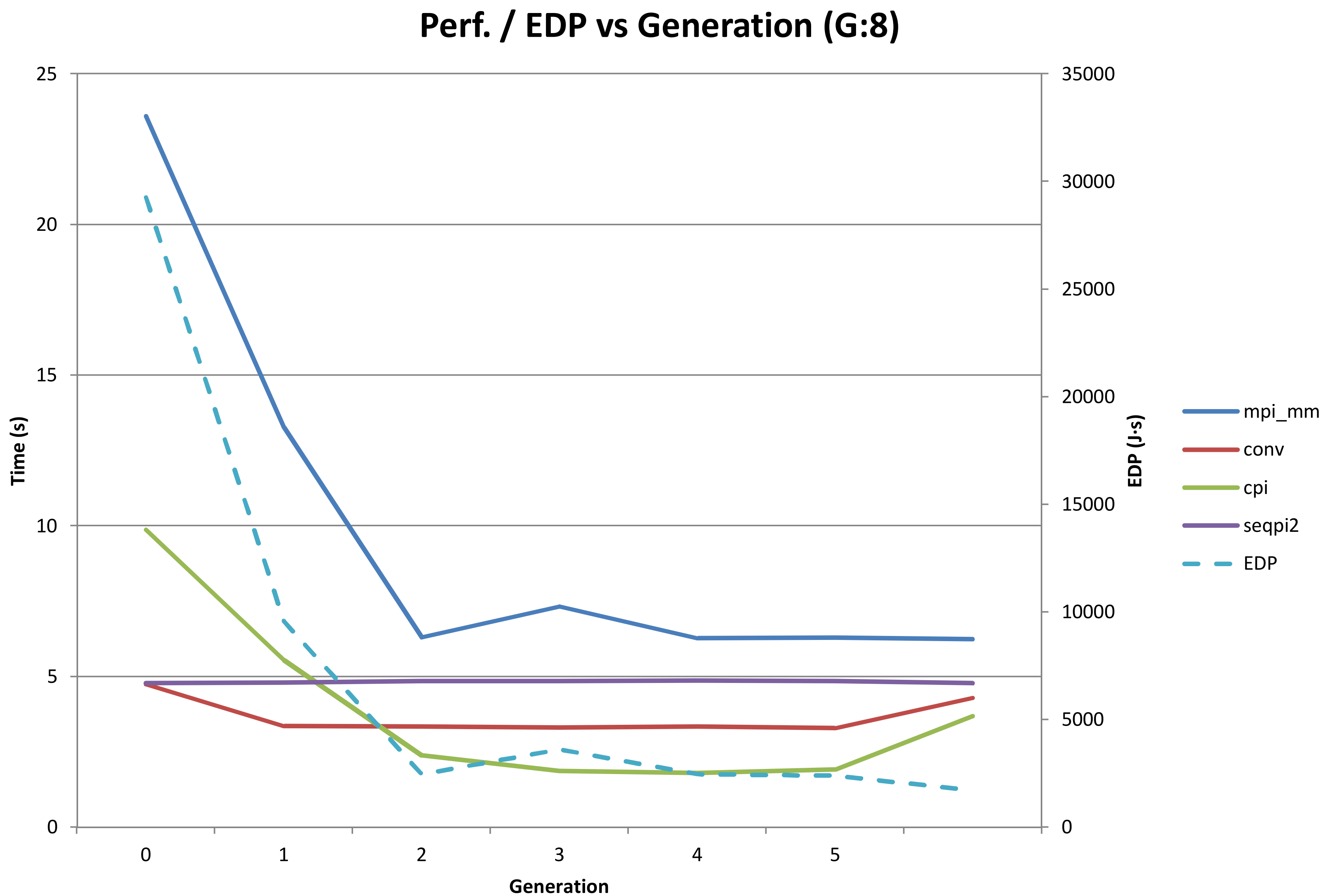
Perf. / EDP vs Generation (G:4)
25
20
15
10
5
0
Time (s)
35000
30000
25000
20000
15000
10000
5000
0
EDP (J·s)
0
1
2
3
4
5
Generation
mpi_mm
conv
cpi
seqpi2
EDP

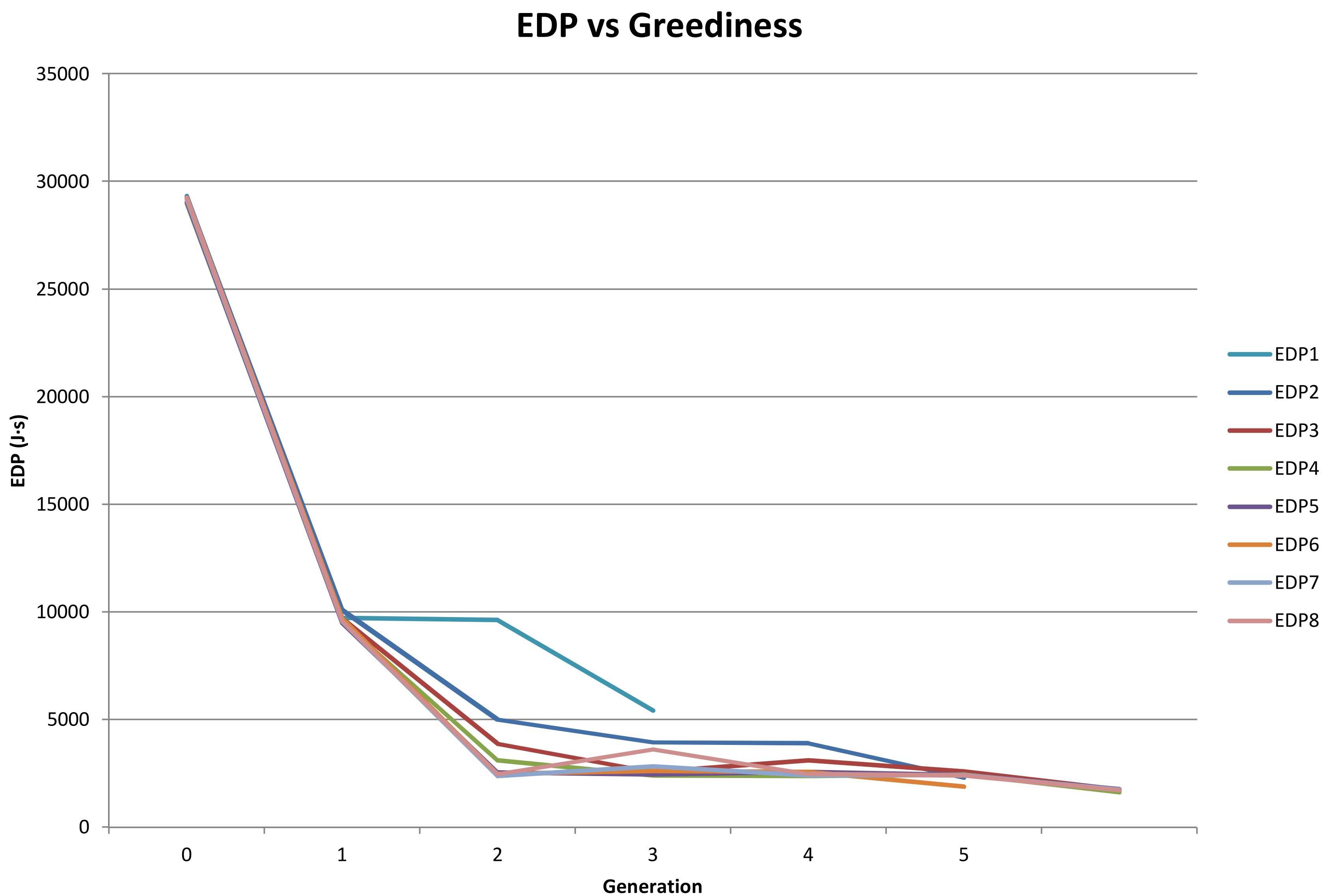

Perf. / EDP vs Generation (G:5)
25
20
15
10
5
0
Time (s)
35000
30000
25000
20000
15000
10000
5000
0
EDP (J·s)
0
1
2
3
4
5
Generation
mpi_mm
conv
cpi
seqpi2
EDP

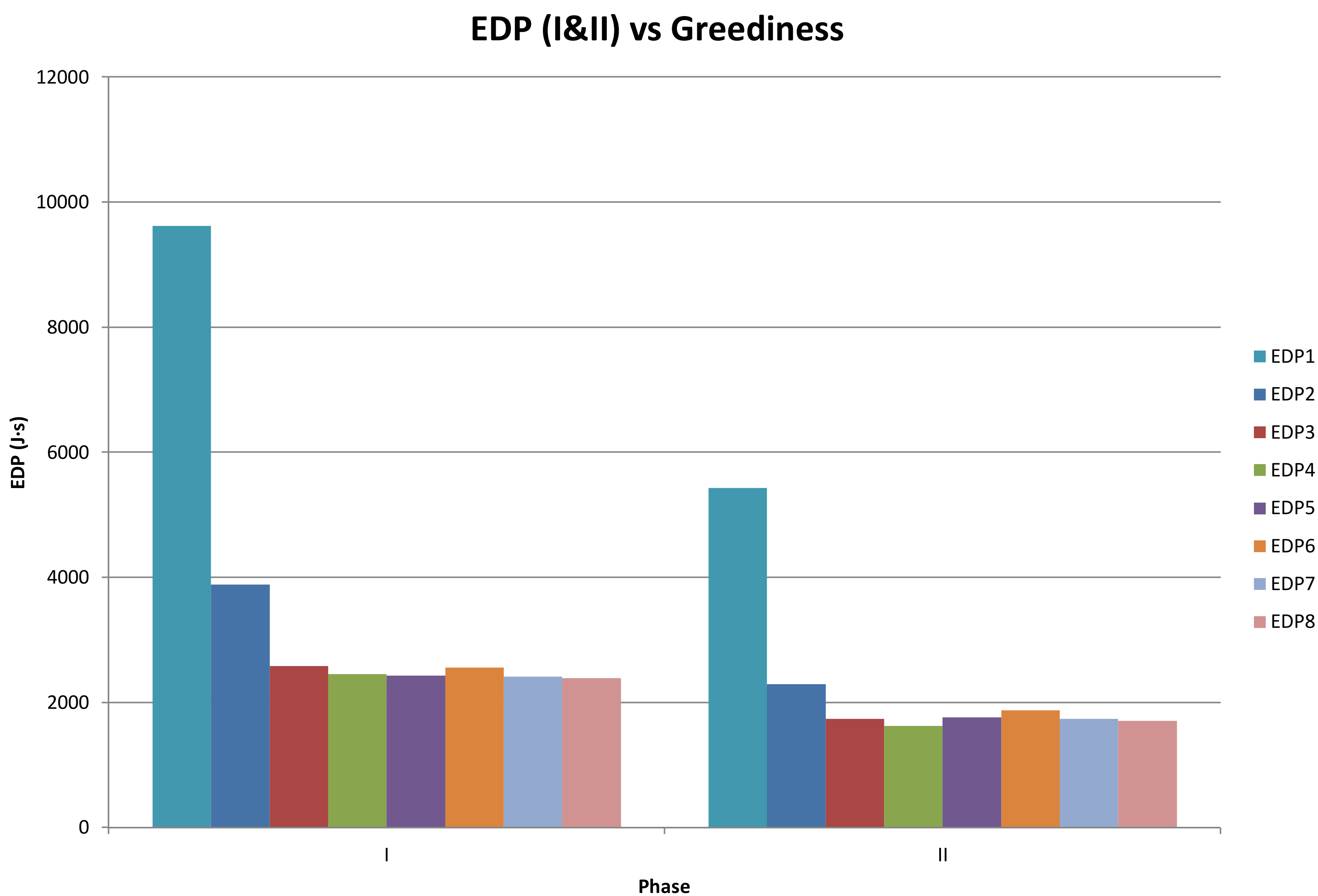

Perf. / EDP vs Generation (G:6)
25
20
15
10
5
0
Time (s)
35000
30000
25000
20000
15000
10000
5000
0
EDP (J·s)
0
1
2
3
4
Generation
mpi_mm
conv
cpi
seqpi2
EDP

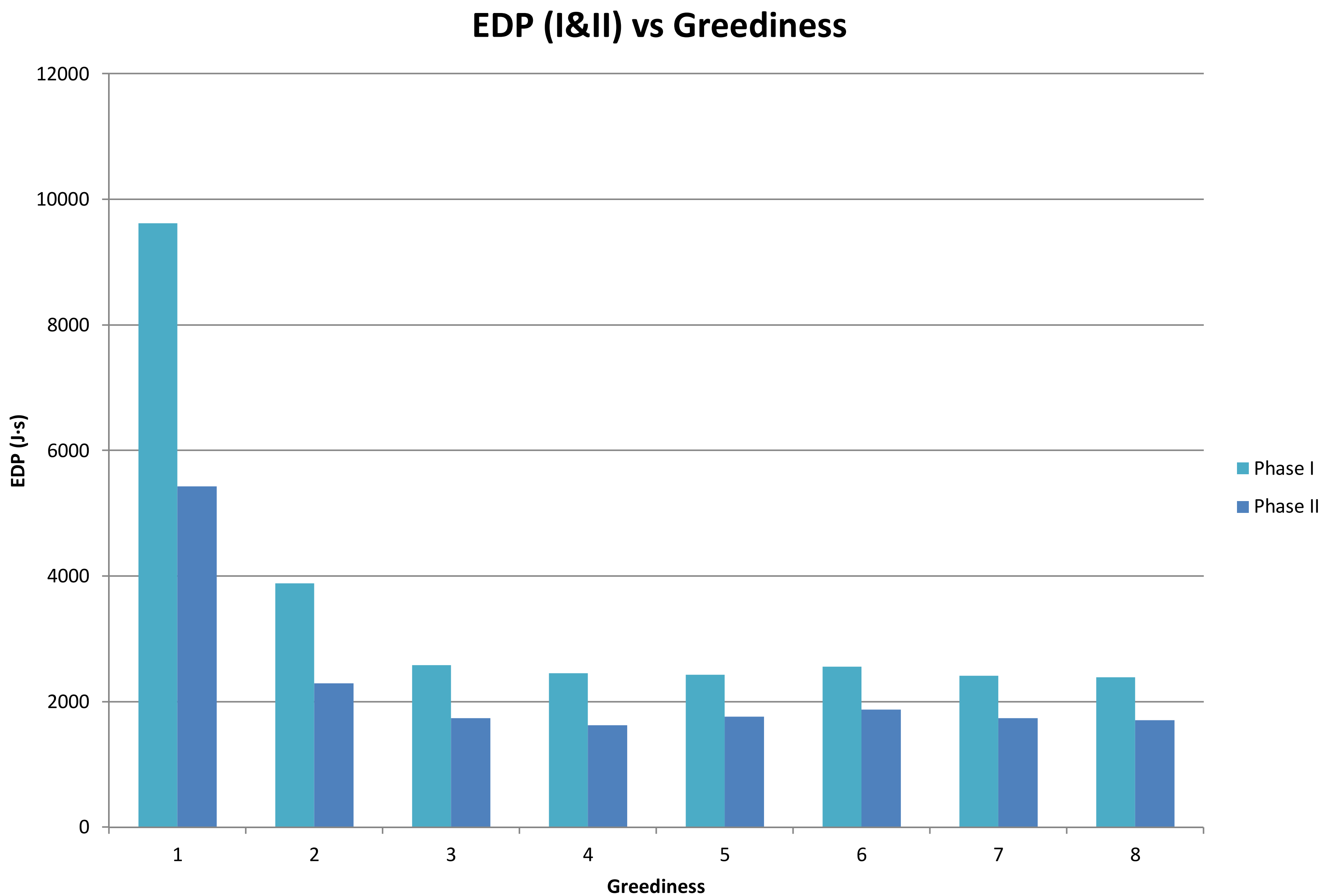
Perf. / EDP vs Generation (G:7)

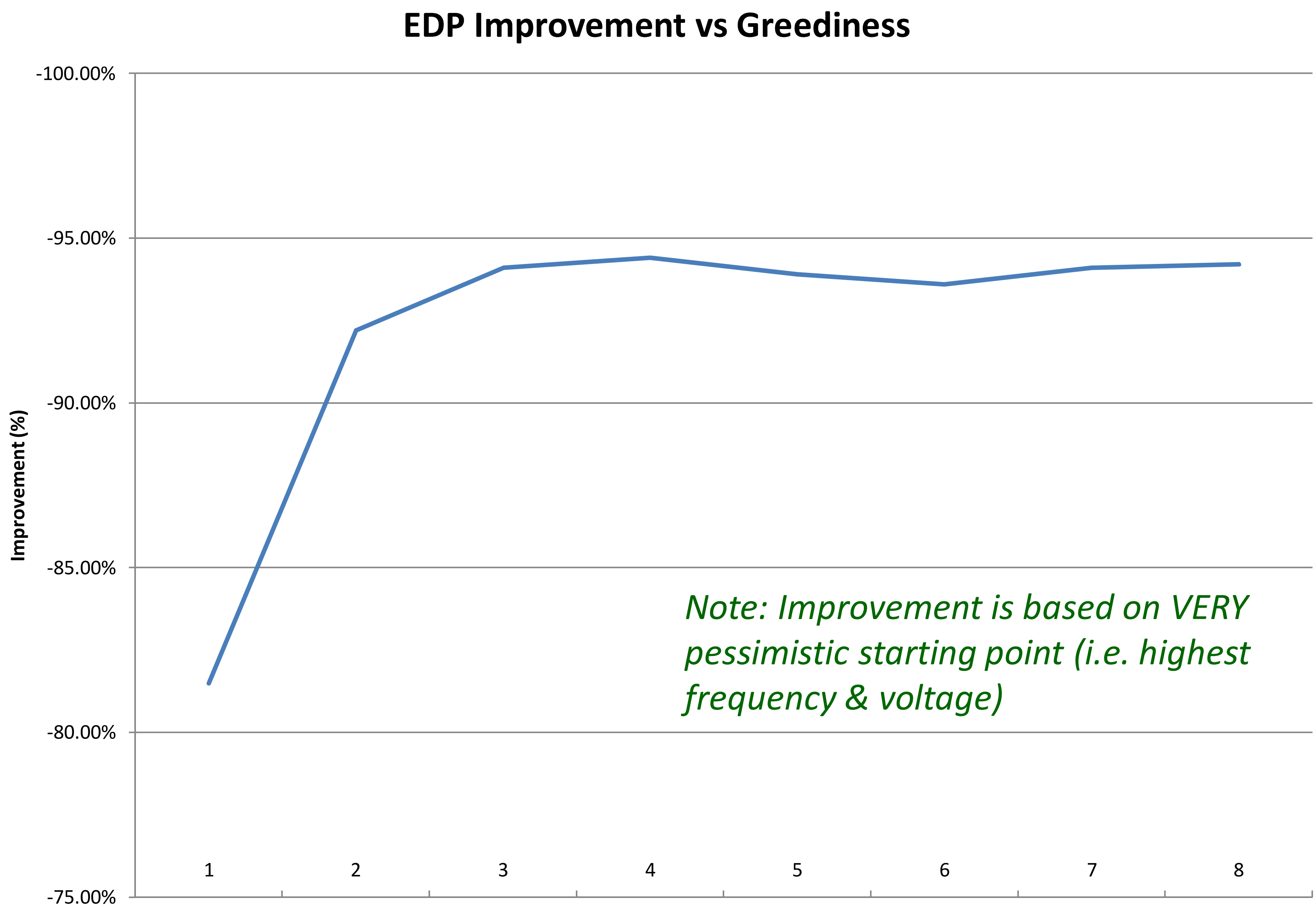



Perf. / EDP vs Generation (G:8)
25
20
15
10
5
0
Time (s)
35000
30000
25000
20000
15000
10000
5000
0
EDP (J·s)
0
1
2
3
4
5
Generation
mpi_mm
conv
cpi
seqpi2
EDP


EDP vs Greediness
EDP (J·s)
35000
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5
Generation
EDP1
EDP2
EDP3
EDP4
EDP5
EDP6
EDP7
EDP8


EDP (I&II) vs Greediness
EDP (J·s)
12000
10000
8000
6000
4000
2000
0
I
II
Phase
EDP1
EDP2
EDP3
EDP4
EDP5
EDP6
EDP7
EDP8


EDP (I&II) vs Greediness
EDP (J·s)
12000
10000
8000
6000
4000
2000
0
1
2
3
4
5
6
7
8
Greediness
Phase I
Phase II

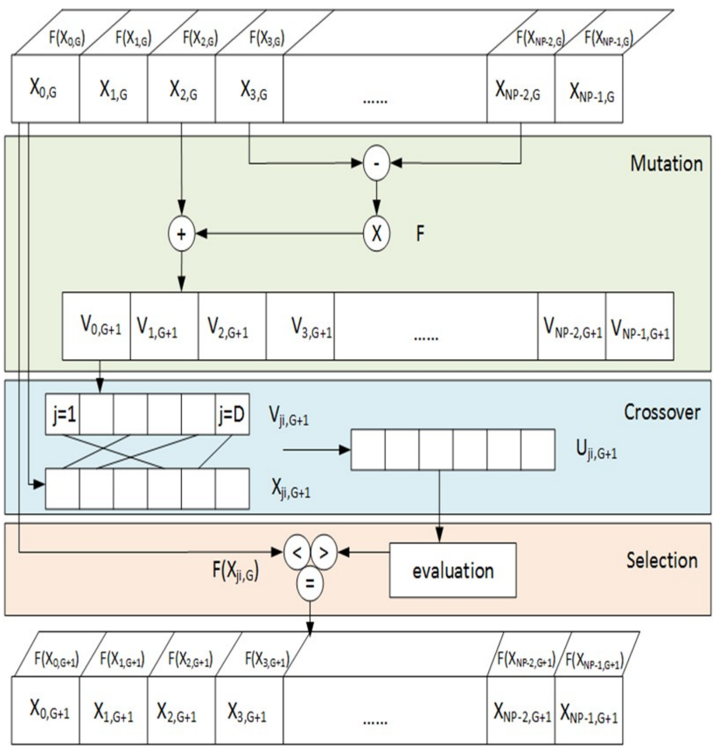

EDP Improvement vs Greediness
-100.00%
-95.00%
-90.00%
-85.00%
-80.00%
-75.00%
Improvement (%)
1
2
3
4
5
6
7
8
Note: Improvement is based on VERY pessimistic starting point (i.e. highest frequency & voltage)

Results Summary

Greediness	Gens	mpi_mm		conv		cpi		seqpi2		Energy(J)	EDP (J·s)	Total Time(s)
		Time (s)	CPUs:Gear	Time (s)	CPUs:Gear	Time (s)	CPUs:Gear	Time (s)	CPUs:Gear			
Phase I												
1	3	13.33	3:1	3.60	2:1	44.82	2:1	4.79	1:1	2,360	105.7 K	178
2	5	8.33	5:1	3.30	3:1	22.99	4:1	4.78	1:1	1,244	28.6 K	210
3	6	6.59	8:1	3.35	3:1	13.70	7:1	4.82	1:1	794	10.9 K	200
4	7	6.31	10:1	3.32	4:1	8.00	13:1	4.81	1:1	504	4,030.3	187
5	8	6.30	11:1	3.35	4:1	6.30	18:1	4.80	1:1	469	2,879.5	185
6	7	6.72	14:1	3.26	5:1	5.95	19:1	4.79	1:1	473	3.182.0	175
7	7	6.38	9:1	3.26	5:1	5.31	22:1	4.87	1:1	453	2,894.8	172
Phase II												
1	4	28.89	3:4	3.28	2:1	44.80	2:1	4.79	1:1	1,314	58.9 K	223
2	6	15.78	5:4	5.26	3:3	23.29	4:1	4.81	1:1	774	18.0 K	223
3	7	12.30	8:4	5.29	3:3	13.67	7:1	4.83	1:1	491	6.7 K	213
4	8	7.78	10:2	6.45	4:3	7.94	13:1	4.80	1:1	390	3093.5	195
5	9	6.26	11:1	5.26	4:2	6.26	18:1	4.86	1:1	416	2,606.1	191
6	8	6.38	14:1	4.28	5:2	5.91	19:1	4.81	1:1	452	2,883.0	181
7	8	6.29	9:1	4.30	5:2	7.82	22:1	4.75	1:1	426	3,333.9	180

SVM-JADE

Fitness competition between the optimum obtained from SVM regression and the worst individual of the current generation, in order to accelerate the convergence process of searching.



SVM module Inserted

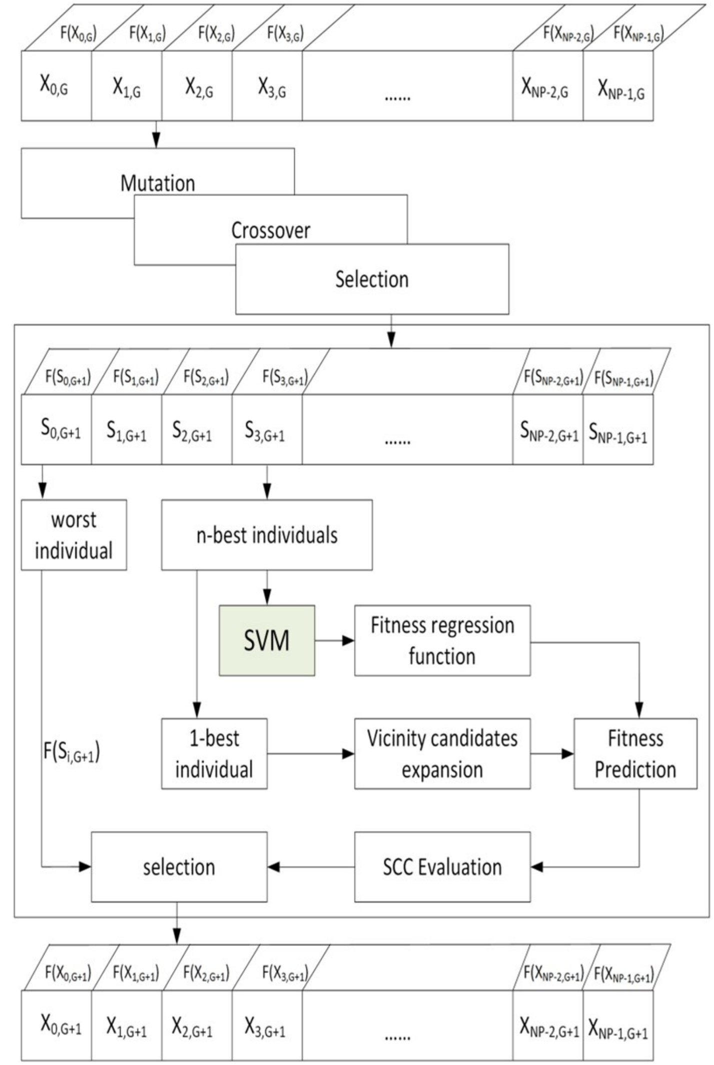


Diagram of traditional Differential Evolutions

Diagram of SVM JADE

Comparison w/Previous Work

Method	Gen	Gear:CPUs				Energy (J)	EDP (J·s)	Algorithm Time (s)
		mpi_mm	conv	cpi	seqpi2			
Jade	9	1:10	2:2	4:20	2:4	428.6	3,277.5	10,320
SVM-Jade	9	1:9	4:16	1:17	1:1	392.9	3,024.5	11,190
Greedy Talents	9	1:11	2:4	1:18	1:1	416.0	2,606.1	191

Conclusions / Contribution

- GreedyTalents uses a simple heuristic
  - Greed (& ambition)
  - can find the same | similar number of processors & gear setting as differential evolution algorithms

- GreedyTalents' simple heuristic makes this determination 50X+ times faster than previous differential evolution auto-tuning algorithms

- Thanks!

  for attending the talk