Evaluation of Chapel's Task Parallel Features

CUG 2009 Atlanta

Dr Michèle Weiland
Applications Consultant, EPCC
m.weiland@epcc.ed.ac.uk

Outline

- Chapel & its task parallel features
- Benchmarks
- Hardware
- Performance
  - Chapel v0.7 vs v0.9
  - single locale
  - multi locale
- Conclusions

Chapel

- Cray's new parallel programming language
  - part of the Cascade project
  - funded through HPCS programme
- increase productivity through
  - built-in high-level support for parallelism
  - support for modern programming paradigms
  - multi-resolution design
- current compiler version 0.9
  - released on 16th April 2009

Task parallel features

- `begin`
  - spawns new parallel task, continues execution immediately
- `sync`
  - waits for completion of dispatched `begin` statements
- `cobegin`
  - compound `begin/sync`
- `coforall`
  - task parallel for loop, guarantees parallel dispatch
- `serial`
  - disables spawning of parallel tasks
- `sync var` & `single var`
  - carry extra state: *full* or *empty*
  - control read/write access

Benchmarks

- Microbenchmarks
  - 5 different implementations to call function 8 times
- Single locale benchmarks – compared to C & Pthreads
  - N-Queens
  - Strassen’s matrix multiplication
  - Mandelbrot set
- Multi locale benchmarks – compared to C & MPI
  - Pi approximation
  - Black-Scholes algorithm

Ness – Sun X4600 compute servers

- **Hardware**
  - Processors: 2.6 GHz dual core AMD Opteron with 2 GB of memory
  - Front-end: 2 processors
  - Back-end: 32 processors (divided into two 16 processors shared memory nodes)

- **Software**
  - Linux OS (Scientific Linux)
  - Sun Grid Engine
  - Compilers
    - GNU 4.1.1
    - PGI 7.0.7

HPCx – IBM eServer 575 cluster

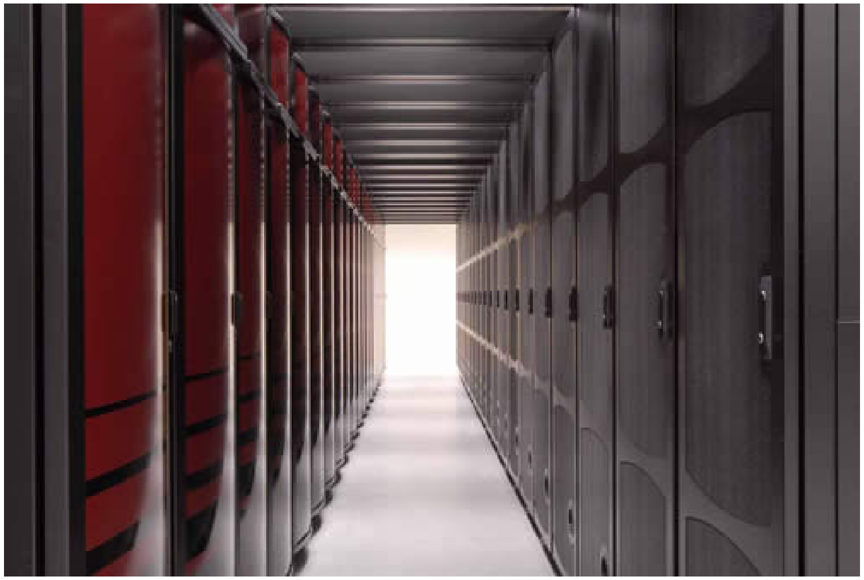
- **Hardware**
  - 2560 cores: 1.5 GHz Power5
  - 8 dual-core chips & 32 GB memory per node
  - "*Federation*" High Performance switch
  - Simultaneous Multi Threading

- **Software**
  - AIX5.3
  - POE & LoadLeveler
  - Compiler
    - XL 8.0

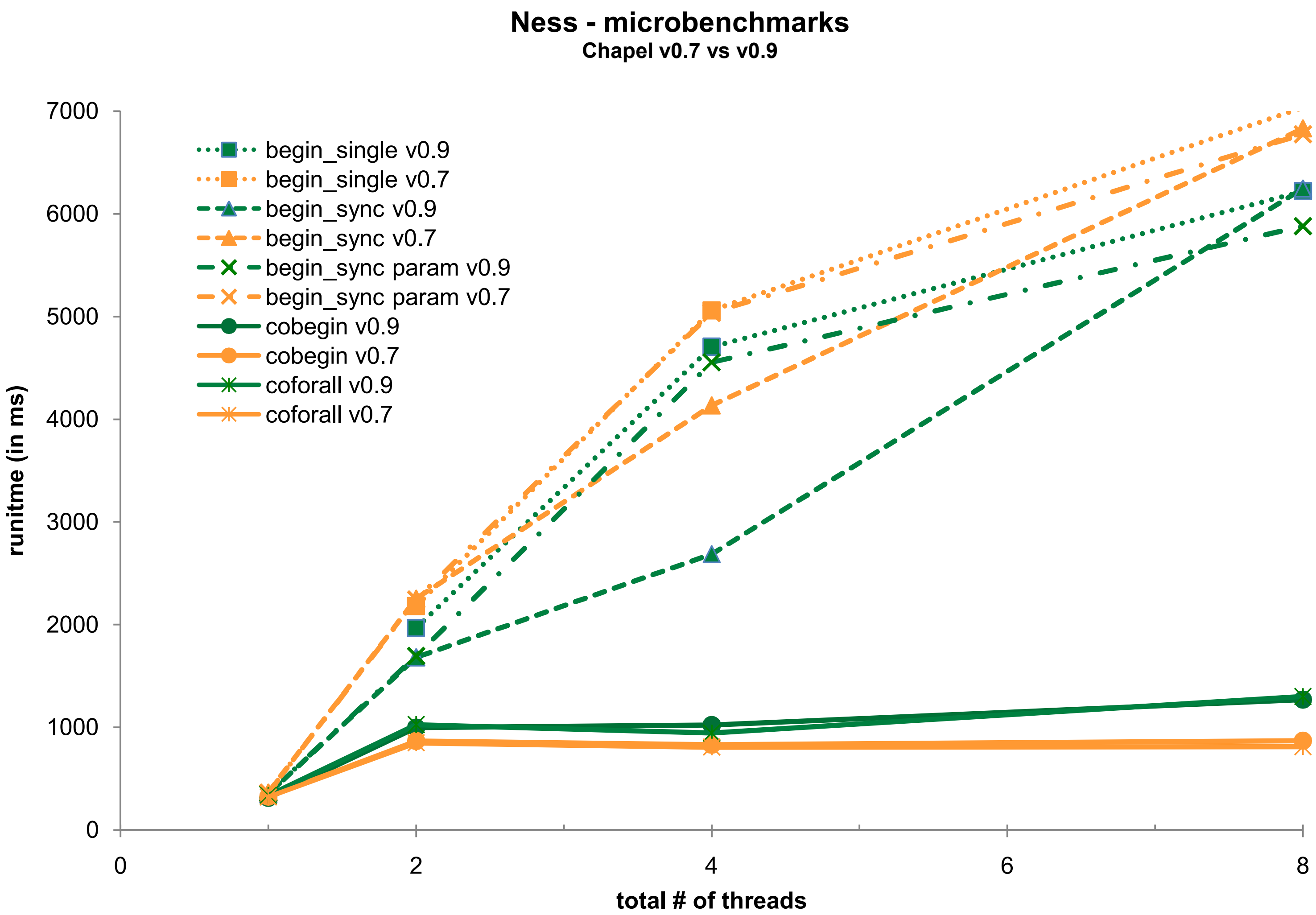
HECToR – Cray XT4

- **Hardware**
  - 11328 cores: 2.8 GHz dual core AMD Opteron chips
  - 6GB memory per chip
  - SeaStar2

- **Software**
  - CLE 2.0.26
    - TDS: CLE 2.1.50HD
  - Compilers
    - GNU 4.1.2
    - PGI 8.0.3



Microbenchmarks

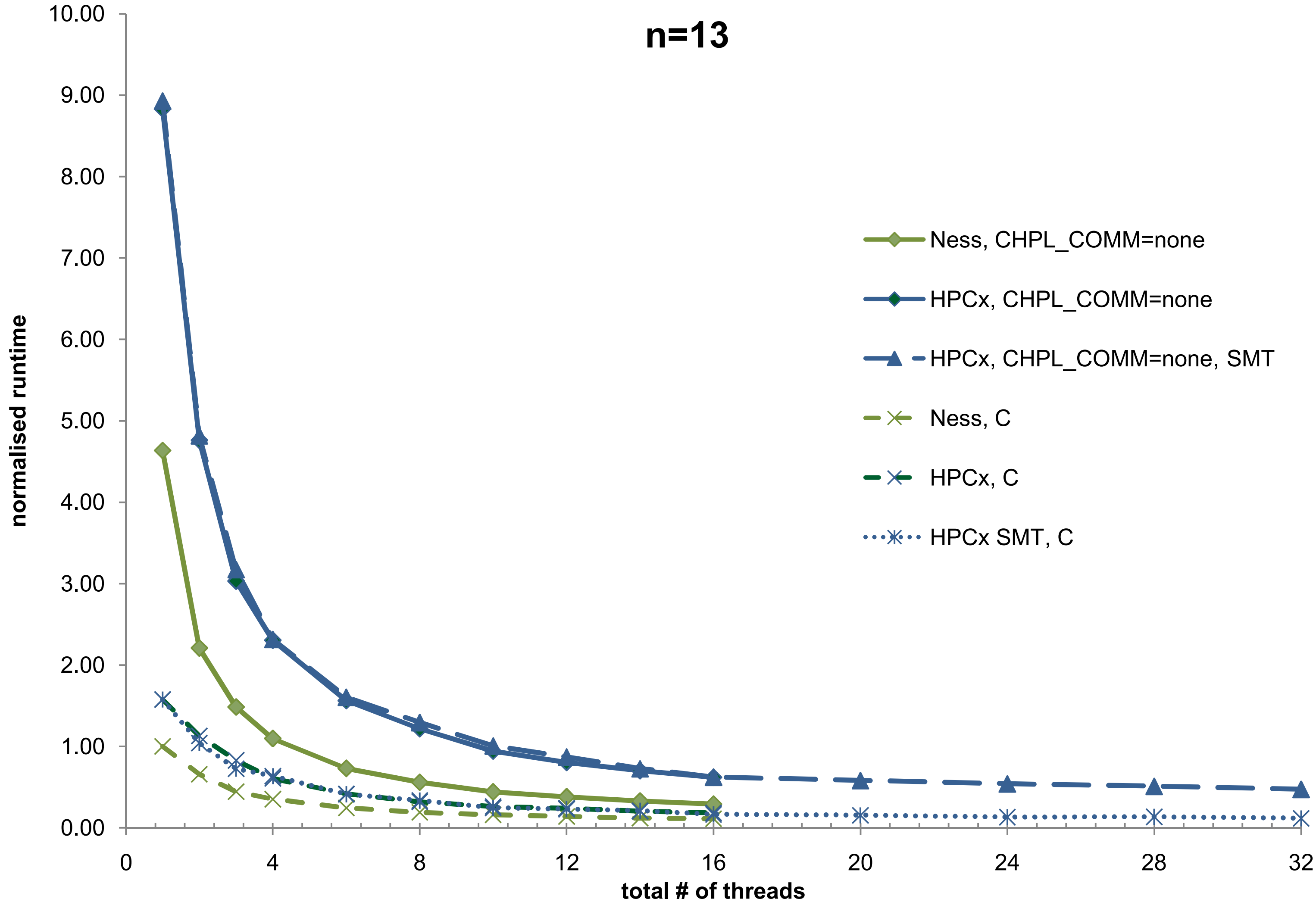


N-Queens benchmark

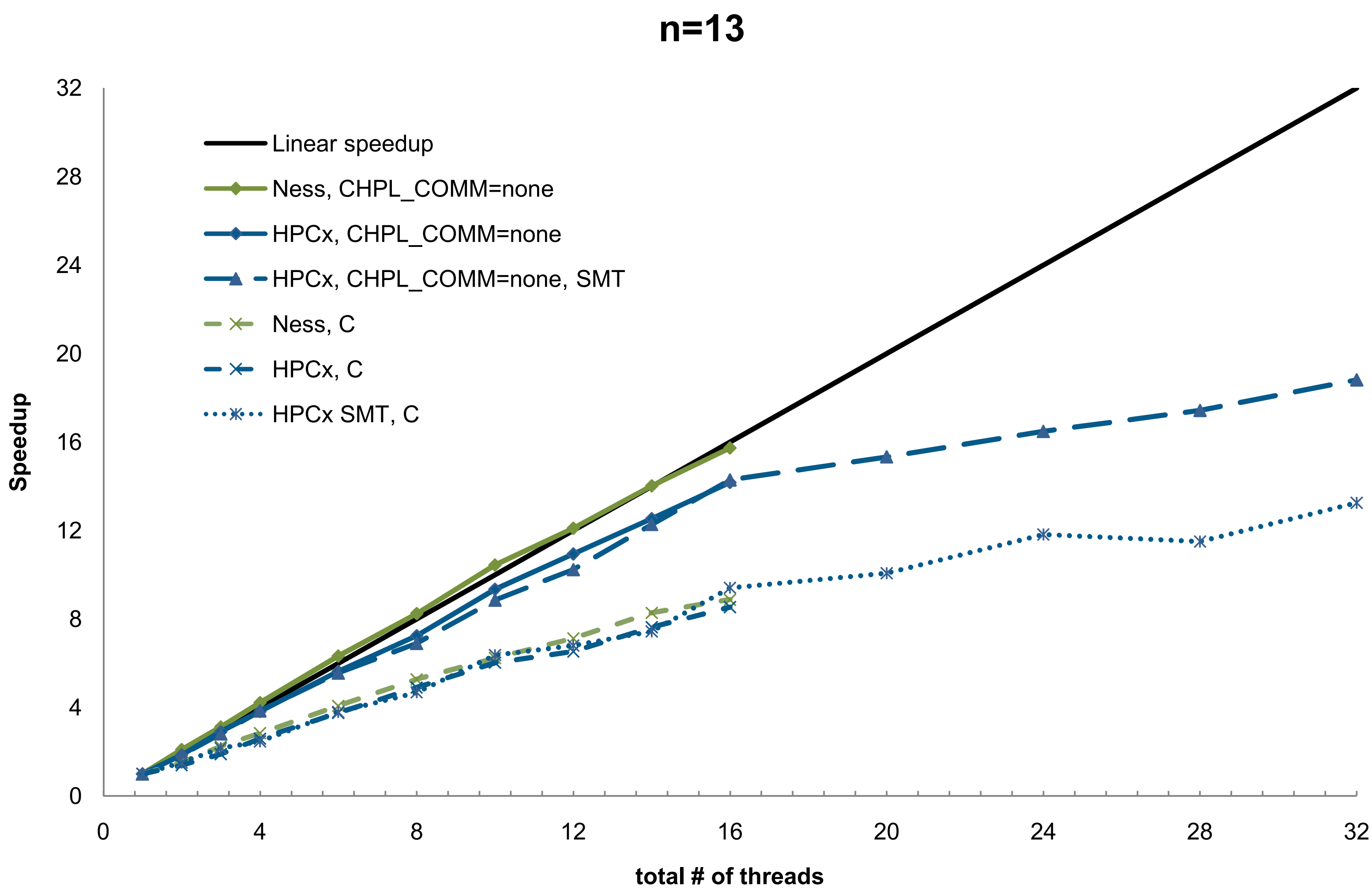
- generalised form of 8 queens puzzle
  - high computational, low memory complexity
- parallelisation
  - spawn parallel tasks for each possible configuration for two rows
    → using `begin` block inside `sync` block

```
sync {
   // for each possible configuration for row 1  and 2 (r1,r2)
   for (r1,r2) in [1..n, 1..n] {
      // if the configuration is safe (i.e. queens do not conflict)
      if( r1!=r2 && r1!=r2+1 && r1!=r2-1) {
         begin {
            // form row 1 and 2 as a configuration array
            var qconfig : [1..n] int;
            qconfig[1..2] = (r1,r2);
            partialSolutions[r1,r2]=nqueens_solver(3,qconfig);
         }
      }
   }
}                                                                          }
```

N-Queens: performance



N-Queens: scaling



Strassen's algorithm

- matrix multiplication algorithm
  - decompose matrices into quadrants, compute 7 new matrices, get result

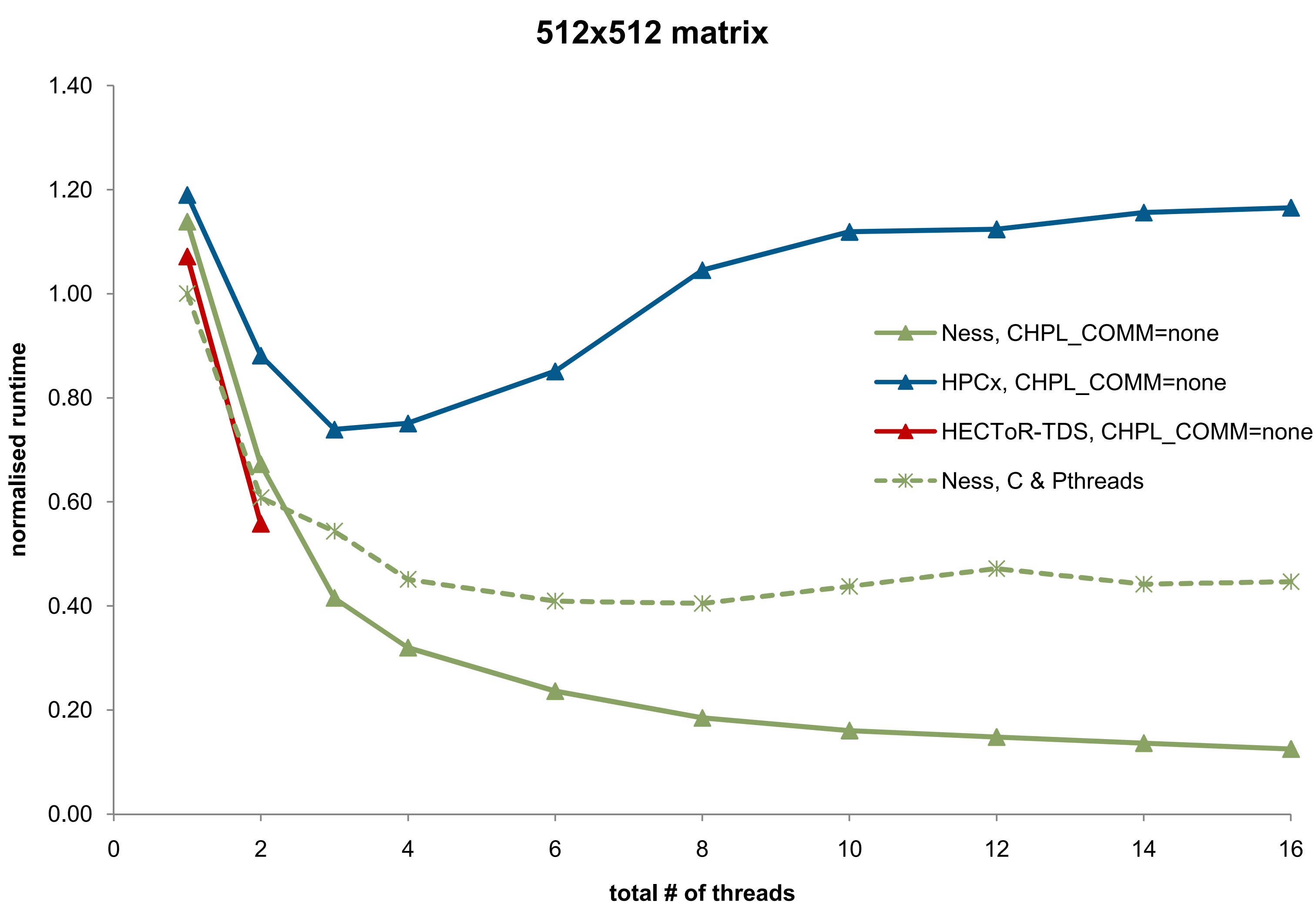
- parallel code uses both domains and whole array operations

- task parallelism using `cobegin` statements:

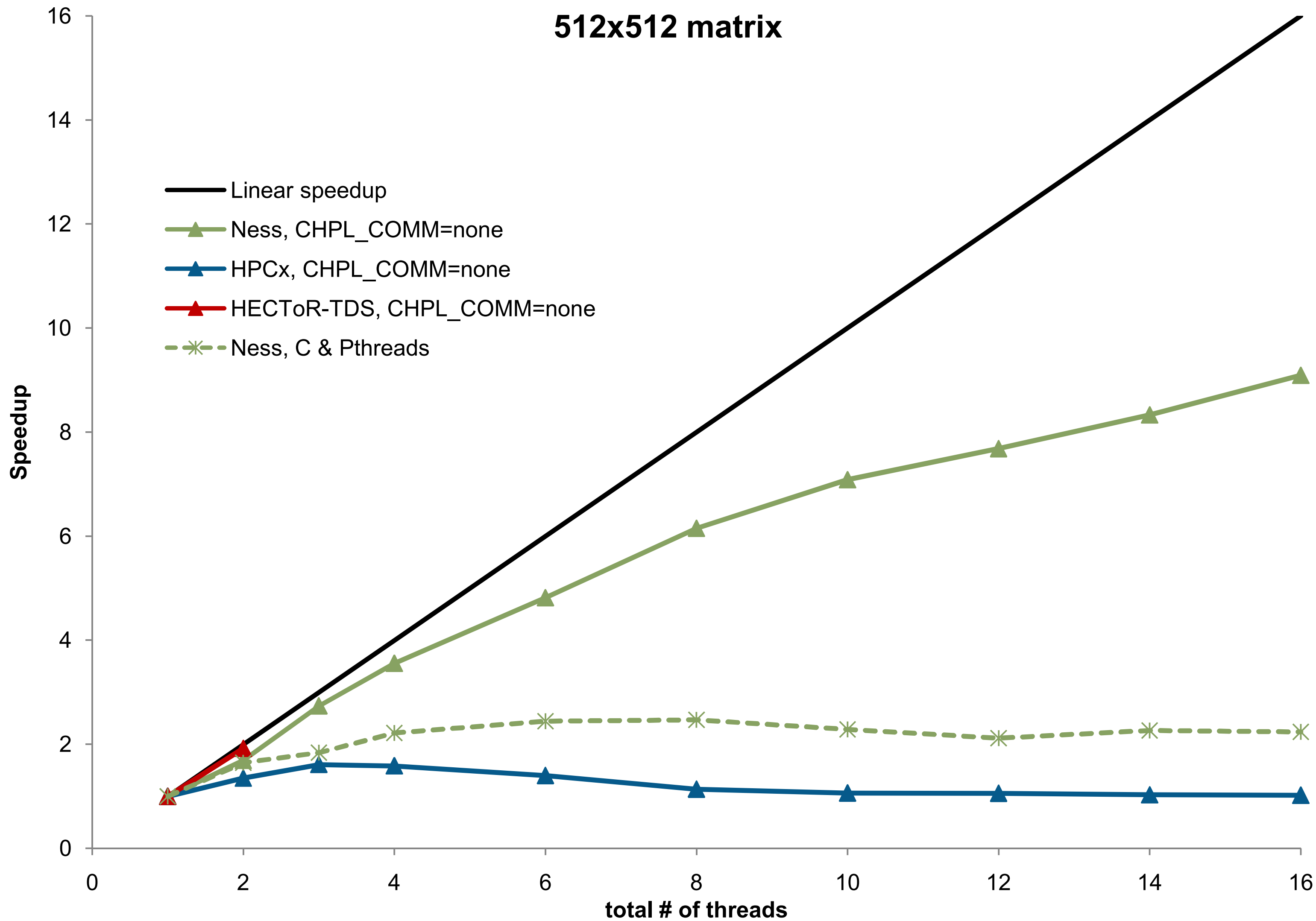
```
cobegin {
   M[1] = MatMul(A11p22,B11p22);
   M[2] = MatMul(A21p22,B[B11]);
   M[3] = MatMul(A[A11],B12m22);
   M[4] = MatMul(A[A22],B21m11);
   M[5] = MatMul(A11p12,B[B22]);
   M[6] = MatMul(A21m11,B11p12);
   M[7] = MatMul(A12m22,B21p22);
}
```

```
cobegin {
   C[Q11] = M[1] + M[4] - M[5] + M[7];
   C[Q12] = M[3] + M[5];
   C[Q21] = M[2] + M[4];
   C[Q22] = M[1] - M[2] + M[3] + M[6];
}
```

Strassen: performance

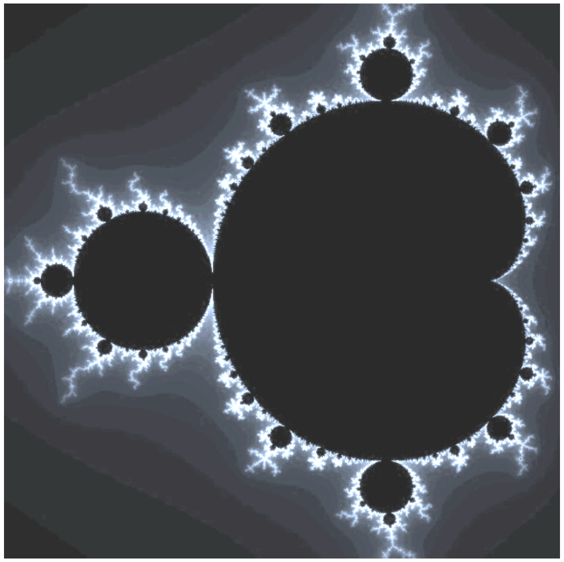


Strassen: scaling



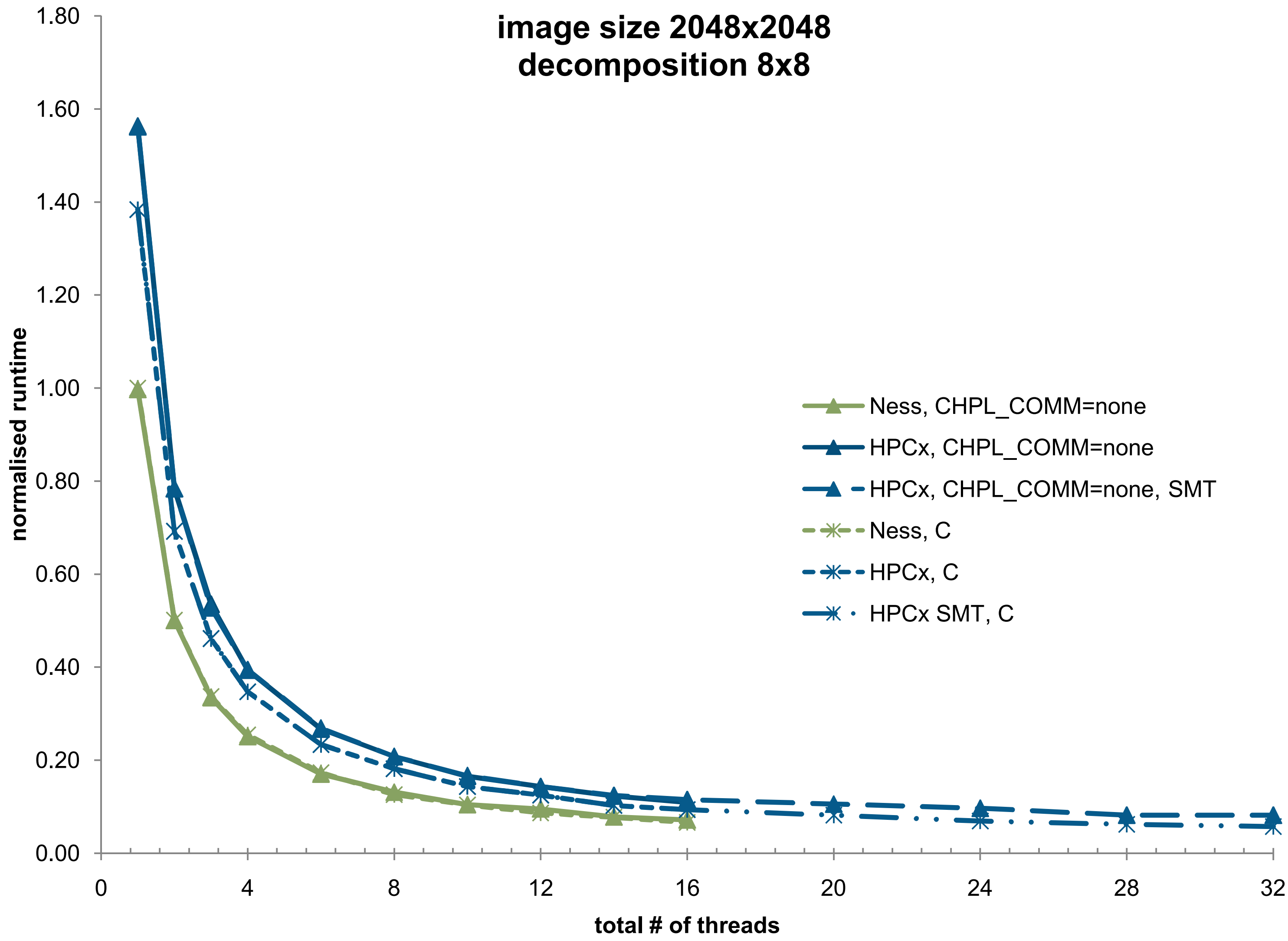
Mandelbrot benchmark

- standard Mandelbrot algorithm
  - 2D array represent image
- parallelism is trivial in Chapel
  - iterator function returns stream of subdomains
  - coforall loops of subdomains in parallel

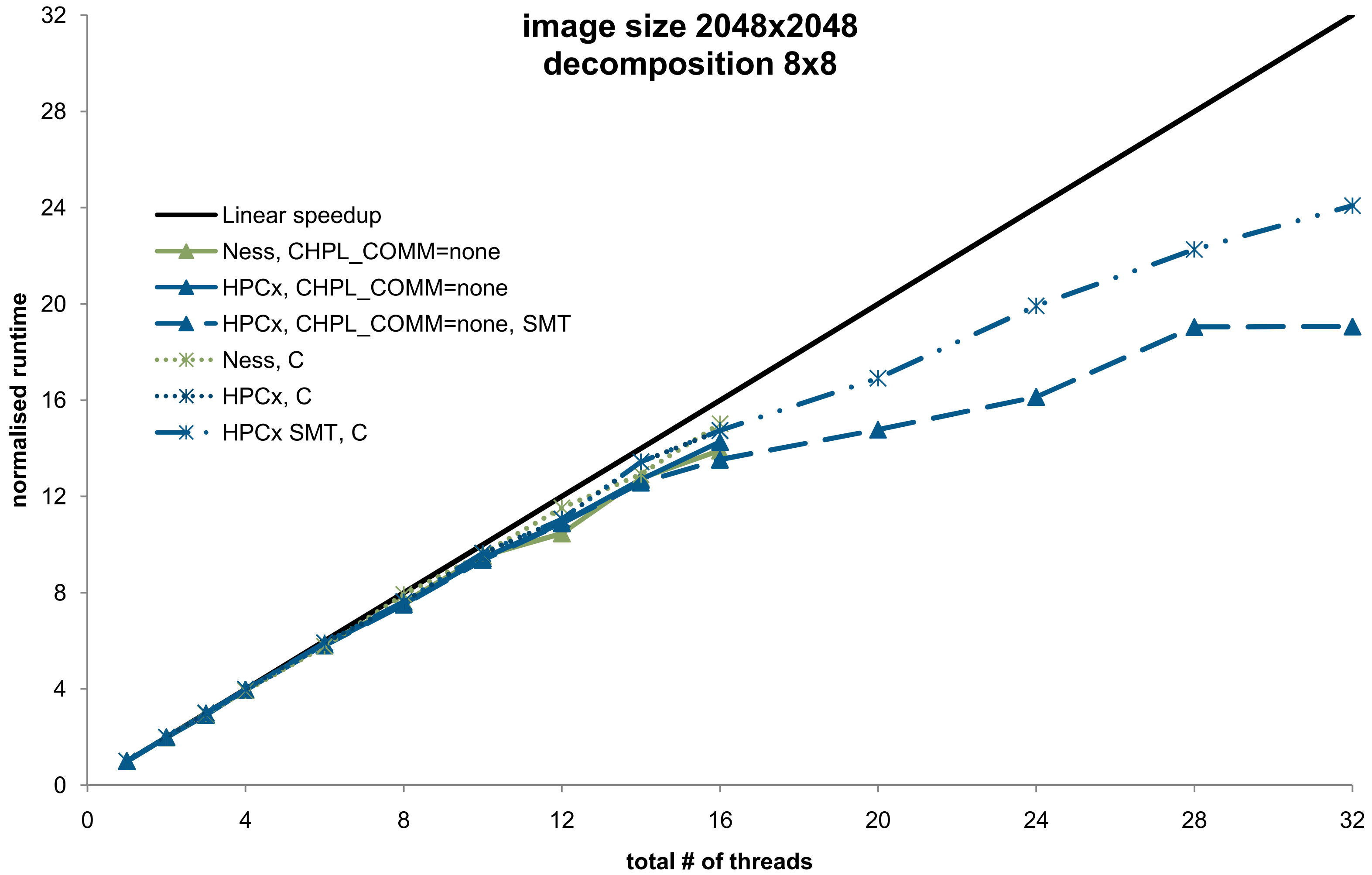


```
coforall d in decomposeDomain(imageD,xdecomp,ydecomp)
        do mandelbrot(image[d]);
```

Mandelbrot: performance



Mandelbrot: scaling



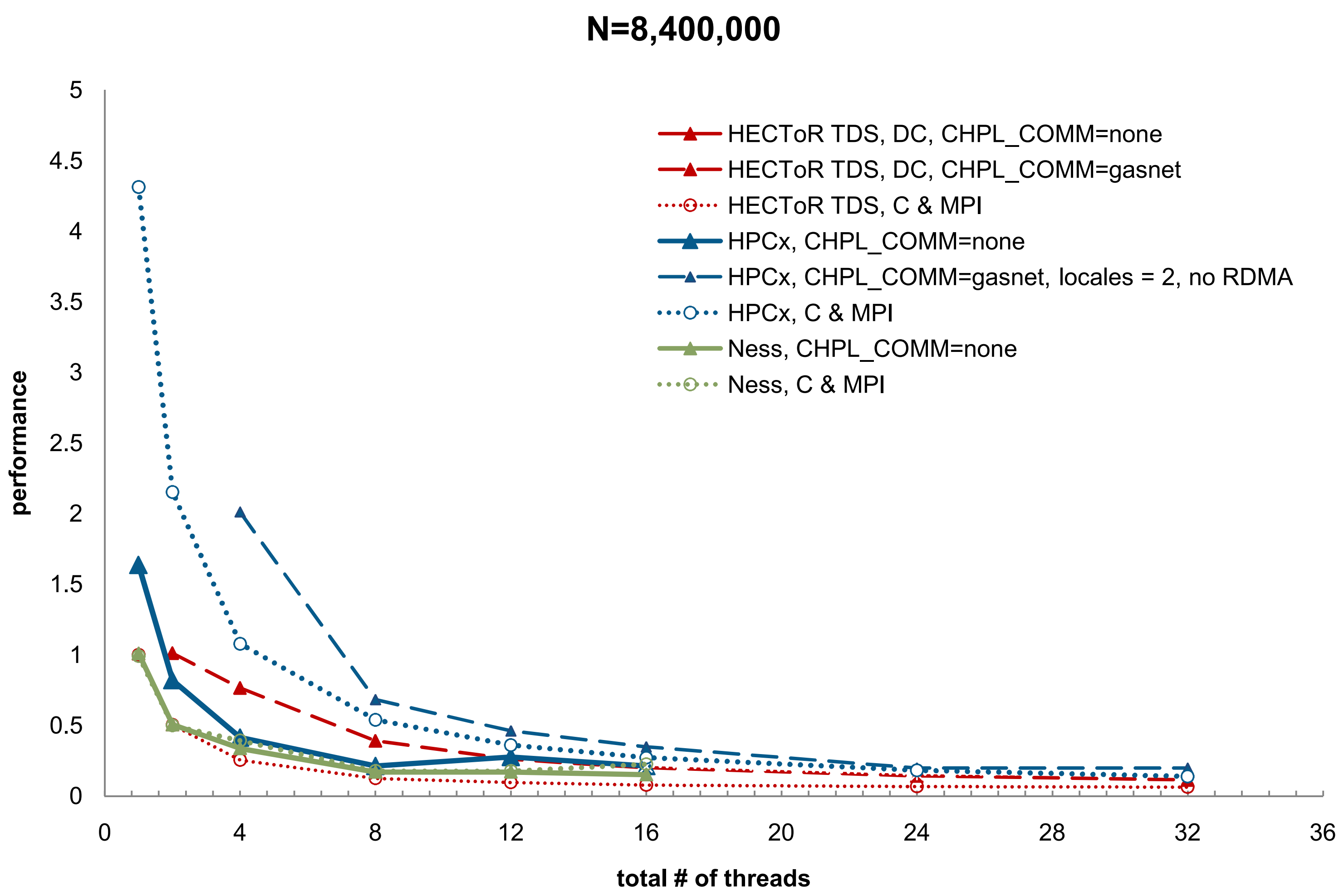
Pi approximation algorithm

- number π can be approximated using

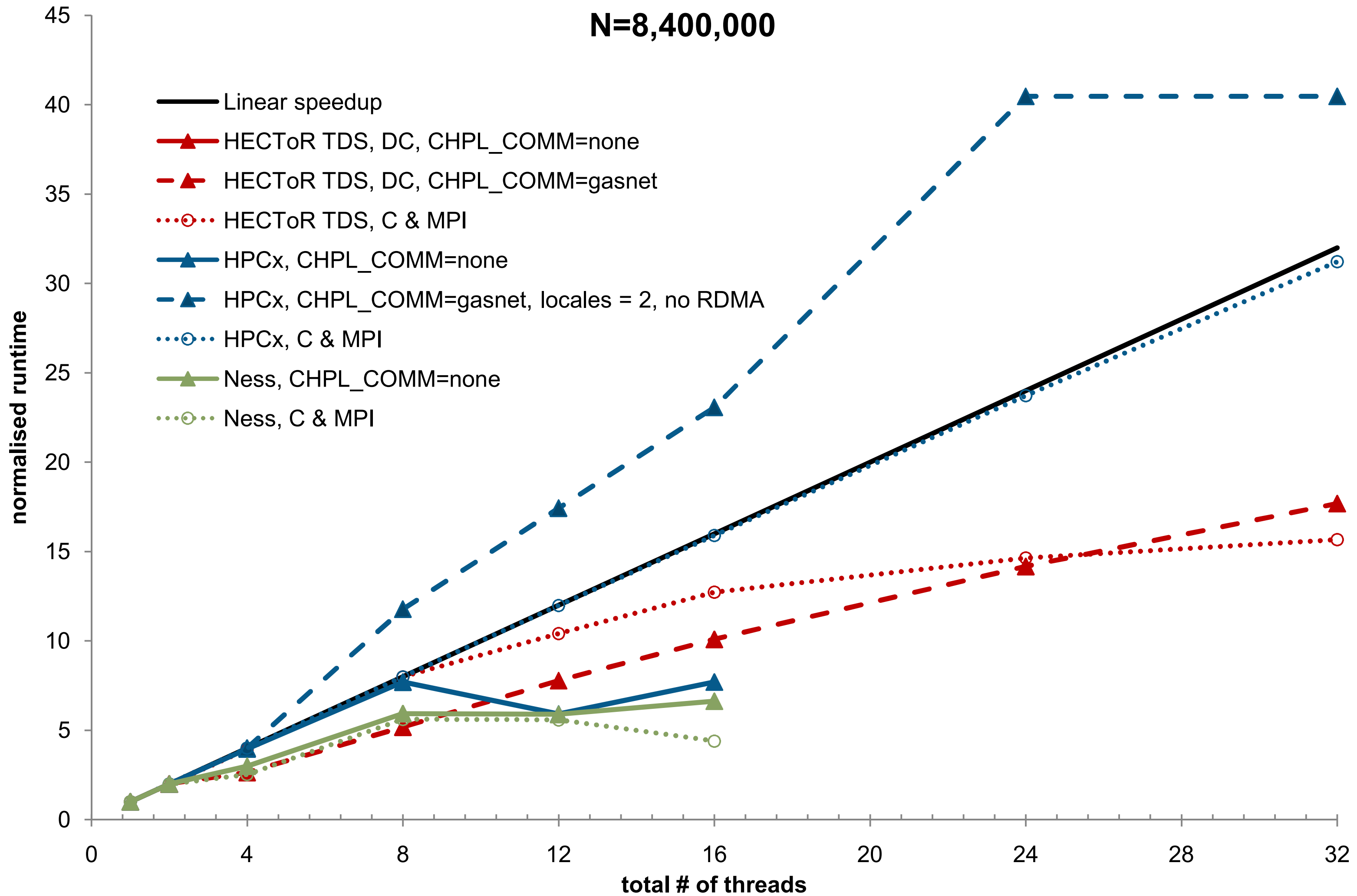
$$\frac{\pi}{4} \approx \frac{1}{N} \sum_{i=1}^{N} \frac{1}{1 + \frac{(i-0.5)^2}{N}}$$

- embarrassingly parallel
  - iterations independent
  - `coforall` loops to distribute over locales and threads
  - `sync var` used to gather partial results

Pi: performance



Pi: scaling


N=8,400,000
Linear speedup
HECToR TDS, DC, CHPL_COMM=none
HECToR TDS, DC, CHPL_COMM=gasnet
HECToR TDS, C & MPI
HPCx, CHPL_COMM=none
HPCx, CHPL_COMM=gasnet, locales = 2, no RDMA
HPCx, C & MPI
Ness, CHPL_COMM=none
Ness, C & MPI
normalised runtime
total # of threads
0
5
10
15
20
25
30
35
40
45
0
4
8
12
16
20
24
28
32

Black-Scholes algorithm

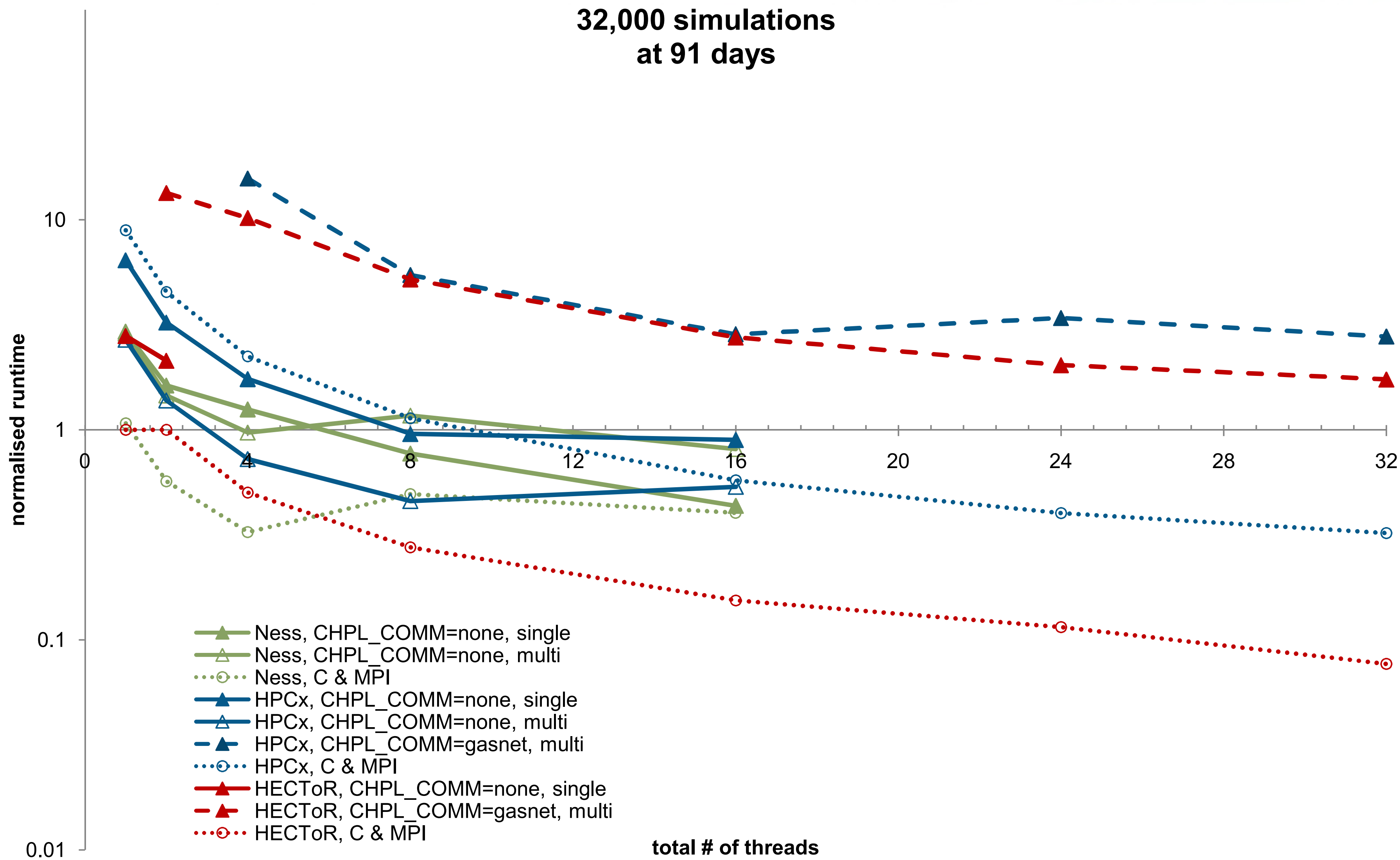
- well-known model from financial theory
  - simulates variation of stock price over time
- real-life application
  - embarrassingly parallel (Monte Carlo technique)
  - simulations are independent of each other
- parallelisation
  - `cobegin` block performs pre-simulation computations
  - 2 `coforall` loops distribute simulations
  - results from simulations are accumulated “locally”
  - `sync vars` used gather final results

serial & single thread performance

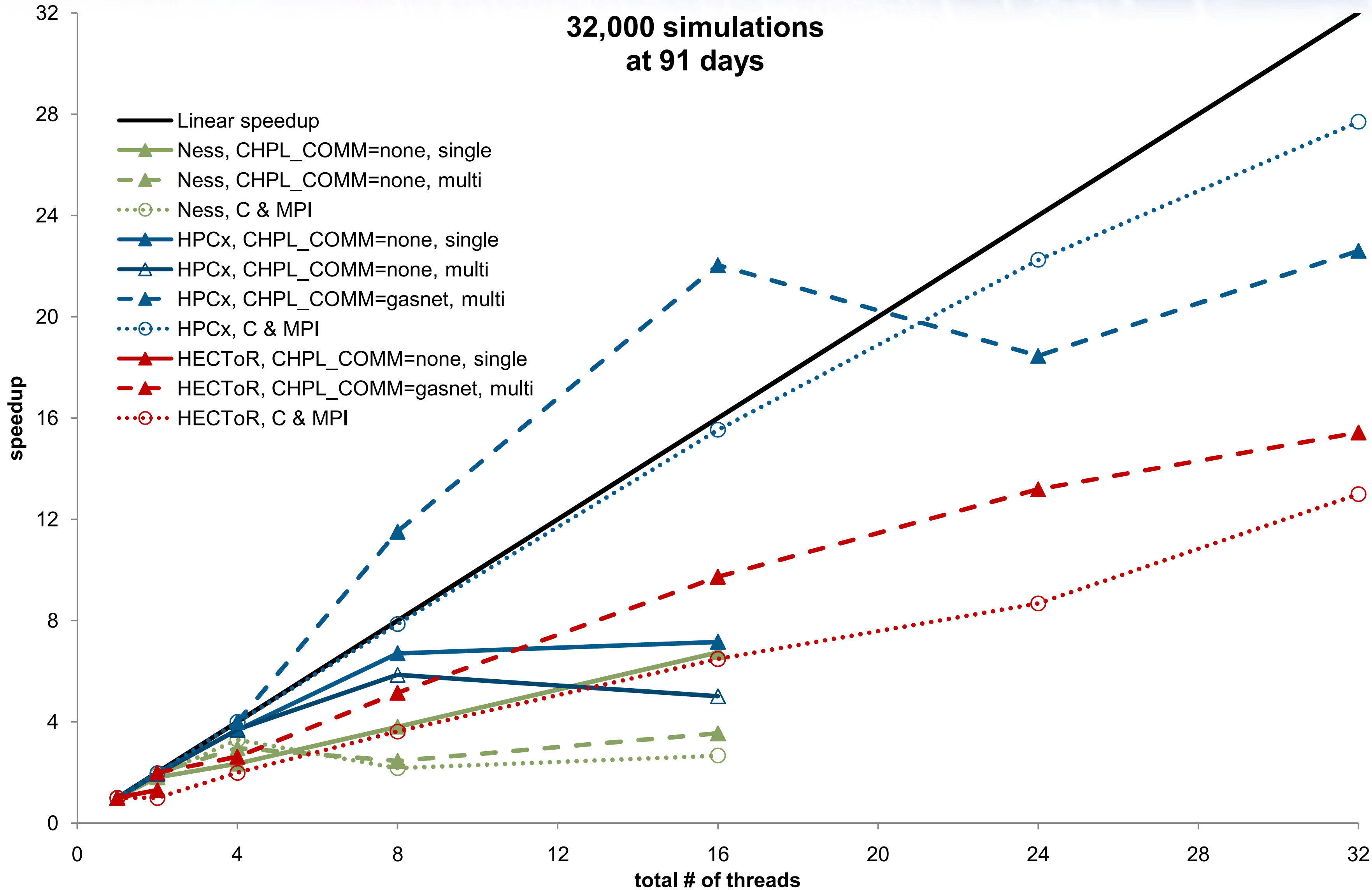
- comparison of serial, single locale and multi locale implementation on 1 thread

	Runtime (ms)
HECToR, C & MPI	9.84
Ness, C & MPI	10.57
HPCx, comm=none, multi	26.40
HECToR, comm=none, single	27.54
Ness, comm=none, multi	28.29
Ness, comm=none, single	28.87
HECToR, serial	31.38
Ness, serial	34.98
HPCx, serial	37.38
HPCx, comm=none, single	63.11
HPCx, C & MPI	87.82

Black-Scholes: performance



Black-Scholes: scaling



Conclusions

- C & Pthreads/MPI often outperform Chapel
  - that's OK!
  - Chapel is work in progress, not optimised heavily
- Encouraging results from Chapel
  - especially on single locales
  - minor issues such as memory leaks remain
- Novel approach to parallel programming with very positive performance picture at this early stage!

Acknowledgement & References

Thanks to the Chapel development team for their continued help and invaluable input!

Chapel: http://chapel.cs.washington.edu/

HPCx: http://www.hpcx.ac.uk

HECToR: http://www.hector.ac.uk