Erlang and Distributed Computing

Colin Morris · Qiyu Zhu

December 2010

Motivation

Erlang is the hot new trend in massively distributed fault-tolerant computation.

In wide current use in applications like Facebook chat and distributed key-value stores.

Does it have anything to offer the world of HPC?

Questions

Is Erlang an appropriate tool for high-performance computing?

Does Erlang really make parallel programming and fault-tolerance easy?

Can we characterize problems that are easy and difficult to make fault-tolerant?

What is Erlang?

The result of Ericsson researchers seeking to design a programming language “that led to the shortest and most beautiful programs closest to the level of formal specifications.”

A concurrent, functional, fault-tolerant, programming language.

Erlang by Example

C

```
X=1;
X=2;
```

Math

```
X = 1
X = 2
```

Erlang

```
X=1.
X=2.
** exception error:
no match of right
hand side value 2
```

Like math, Erlang is single assignment. Once bound, a variable cannot be assigned to again. This is one way Erlang reduces side effects.

Erlang by Example pt. 2

C

```c
int factorial(int n) {
   int result = 1;
   for (int i=1; i<=n; i++){
      result *= i;
   }
   return result;
}
```

Math

$$n! = \begin{cases} 1 & \text{if } n = 0, \\ (n-1)! \times n & \text{if } n > 0. \end{cases}$$

Erlang

```erlang
fact(0) -> 1;
fact(n) -> n*fact(n-1).
```

Erlang performs repetition only by recursion. No loops.

A Refinement

```
fact(n) -> fact(n, 1).

fact(0, Acc) -> Acc;
fact(n, Acc) -> fact(n-1, n*Acc).
```

We use **tail recursion** to avoid growing the stack. We keep a running sum in an accumulator variable Acc, and never have to pass control back up to the calling function.

Erlang by Example pt. 3

C

```c
int j = 0;
for (int i=0; i<sizeof(S)/sizeof(*S); i++){
   if (S[i]%2==0) {
      E[j] = S[i];
      j++;
   }
}
```

Math

$$E = \{x \in S : 2 \mid x\}$$

Erlang

```erlang
E = [X || X <- S, X rem 2 =:= 0]
```

Concurrency in Erlang

Erlang uses asynchronous message passing as its only form of interprocess communication.

No shared memory.

Actor model.

Erlang by Example pt. 4

```
dolphin_server() ->
    receive
        {From, do_a_flip} ->
            From ! "How about no?",
            dolphin_server();
        {From, fish} ->
            From ! "So long and thanks for all the fish!";
        _ ->
            io:format("Heh, we're smarter than you humans.~n"),
            dolphin_server()
   after 10000 ->
      io:format(“Bored now.”)
    end.

1> Dolphin = spawn(dolphins, dolphin_server, []).
<0.75.0>
2> Dolphin ! {self(), do_a_flip}
3> flush().
Shell got “How about no?”
4> Dolphin ! {self(), unknown_message}
Heh, we're smarter than you humans.
```

Example code from “Learn You Some Erlang for Great Good!” (learnyousomeearlang.com)

Fault Tolerance

From the beginning, Erlang was designed to be fault tolerant. In the face of hardware or software errors, a fault-tolerant system should march on.

Erlang encourages "programming for the correct case".

If something goes wrong, the process can just fail and be restarted.

Or can it?

Why does fault tolerance matter?

Modern PCs may run for weeks without needing rebooting. Today's supercomputers often run for only a few days before rebooting, because of their complexity and their thousands of processors. Exascale systems will be even more complex and have millions of processors.... For exascale systems, new fault tolerance paradigms will need to be developed and integrated into both existing and new applications.

"Modeling and Simulation at the Exascale for Energy and the Environment" , 2007: Horst Simon LBNL, Thomas Zacharia ONRL, Rick Stevens ANL

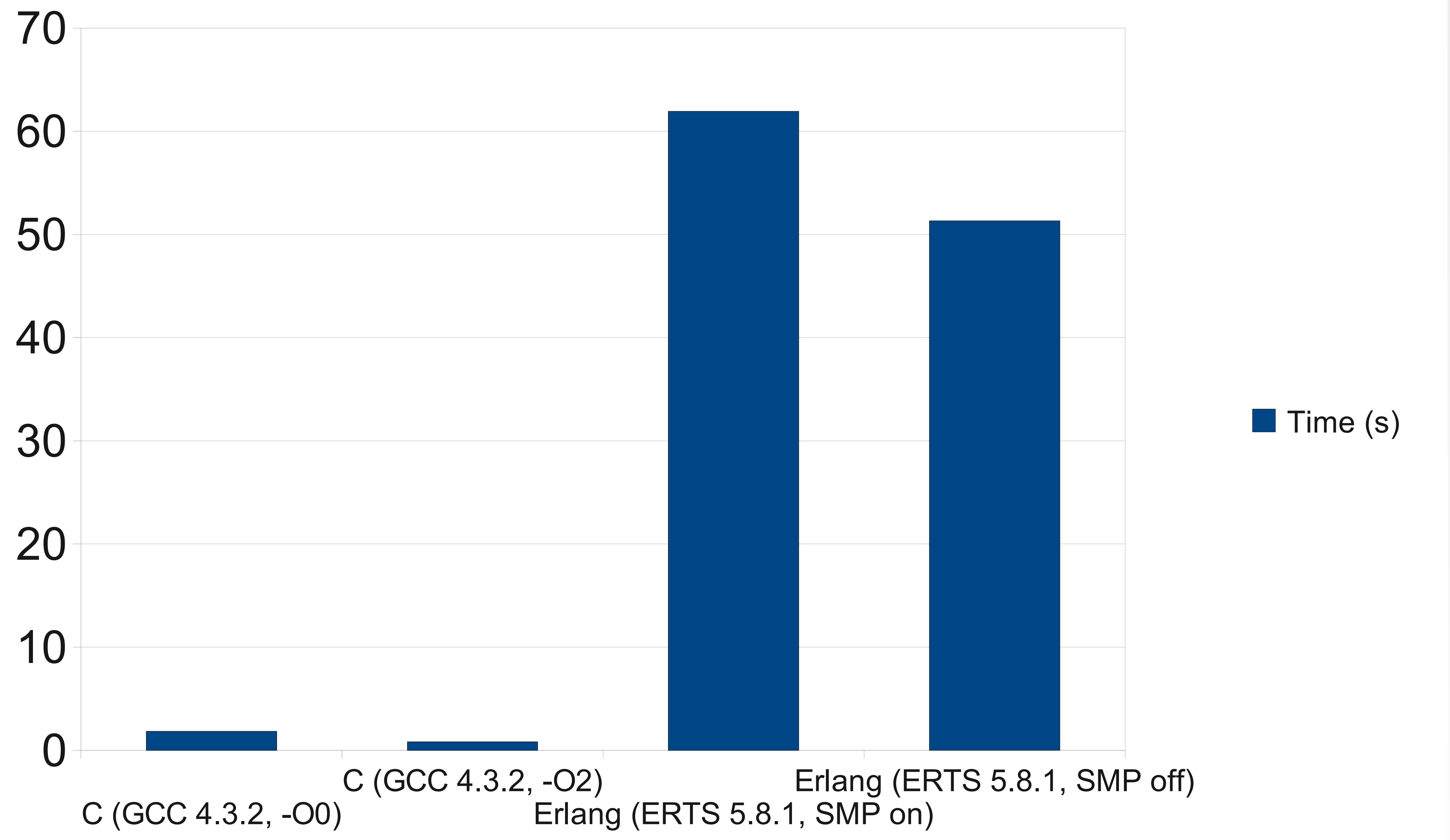
What *doesn't* Erlang do?

Erlang has limited support for arrays, and no support for multi-dimensional arrays.

Erlang is not fast. We found it to be about 50–100 times slower than C for numerical computation.

No numerical libraries.

Compared to C



Case studies

Cowichan Problems

A set of simple computation problems with same properties as real HPC applications

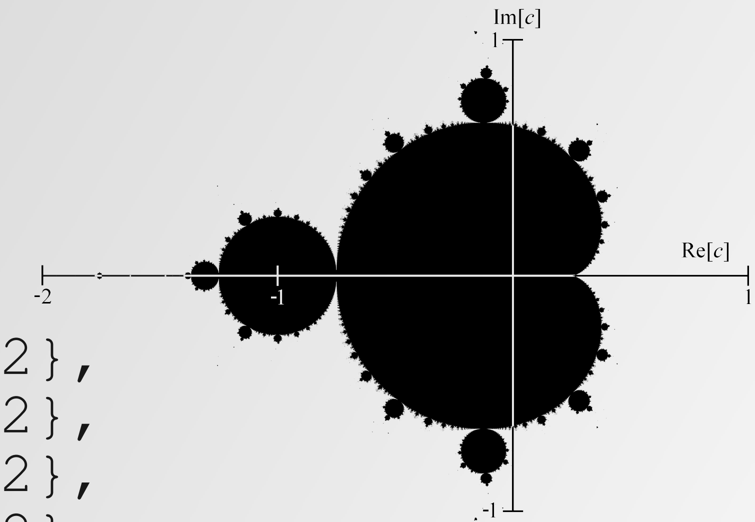
Tests the programmer's ability and the programming environment's capability

Illustrates the practicality of new programming paradigms and parallelization

http://code.google.com/p/cowichan/

Case study: Mandelbrot set

Problem: Generate the Mandelbrot Set for a specified region of the complex plane.



Example:
mandel(10, 10, -1.5, -1.5, 3, 3).

```
{2, 2,  2, 2,  2,  2,  2, 2, 2, 2},
{3, 3,  3, 4,  12, 4,  2, 2, 2, 2},
{3, 3,  4, 6,  150,6,  4, 2, 2, 2},
{4, 5,  7, 150,150,150,6, 3, 2, 2},
{9, 150,22,150,150,150,6, 3, 2, 2},
{9, 150,22,150,150,150,6, 3, 2, 2},
{4, 5,  7, 150,150,150,6, 3, 2, 2},
{3, 3,  4, 6,  150,6,  4, 2, 2, 2},
{3, 3,  3, 4,  12, 4,  2, 2, 2, 2},
{2, 2,  2, 2,  2,  2,  2, 2, 2, 2},
```

The difficult part

Erlang doesn't do arrays.

We wrote a 2D array library using lists of lists.

Erlang

```
lists:sublist(A, j-1) ++
[lists:sublist(lists:nth(j, A),
k-1) ++ [n] ++
lists:sublist(lists:nth(j, A),
k+1, length(A)]
++ lists:sublist(A, j+1,
length(A)).
```

C

```
A[j][k] = n
```

This is supposed to be one line

The easy parts

Embarrassingly parallel: we can just apply a parallel map.

Easy to make fault tolerant: little sense of state. Any one worker process could die and be restarted without affecting any other worker processes.

A natural fit with the functional paradigm.

Fault tolerance in Mandelbrot

```
parallel_mandel(Nrows, Ncols, X0, Y0, Dx, Dy) ->
    Mat = init_matrix(Nrows, Ncols, X0, Y0, Dx, Dy),
    pmap(fun mandel:mandel_number/1, Mat).

% Simple parallel map function
pmap(Fun, Arr) -> Parent = self(),
    Pids = lists:map(fun(Ele) ->
        spawn(fun() -> pmap_f(Parent, Fun, Ele) end) end, Arr),
    pmap_gather(lists:zip(Pids, Arr), Fun).

pmap_f(Parent, Fun, Element) ->
    Parent ! {self(), lists:map(Fun, Element)}.

pmap_gather([{Pid, Elem}|T], Fun) ->
    receive
        {Pid, Ret} -> [Ret|pmap_gather(T)]
    after 2000 ->
        spawn(fun() -> pmap_f(self(), Fun, Elem) end),
        pmap_gather(T ++ [{Pid, Elem}], Fun)
    end;
pmap_gather([], _) -> [].
```

Case study: Game of Life

Small amount of computation per cell

Uniform workload

Peer-to-peer communication

But

Reading neighbor cells is awkward

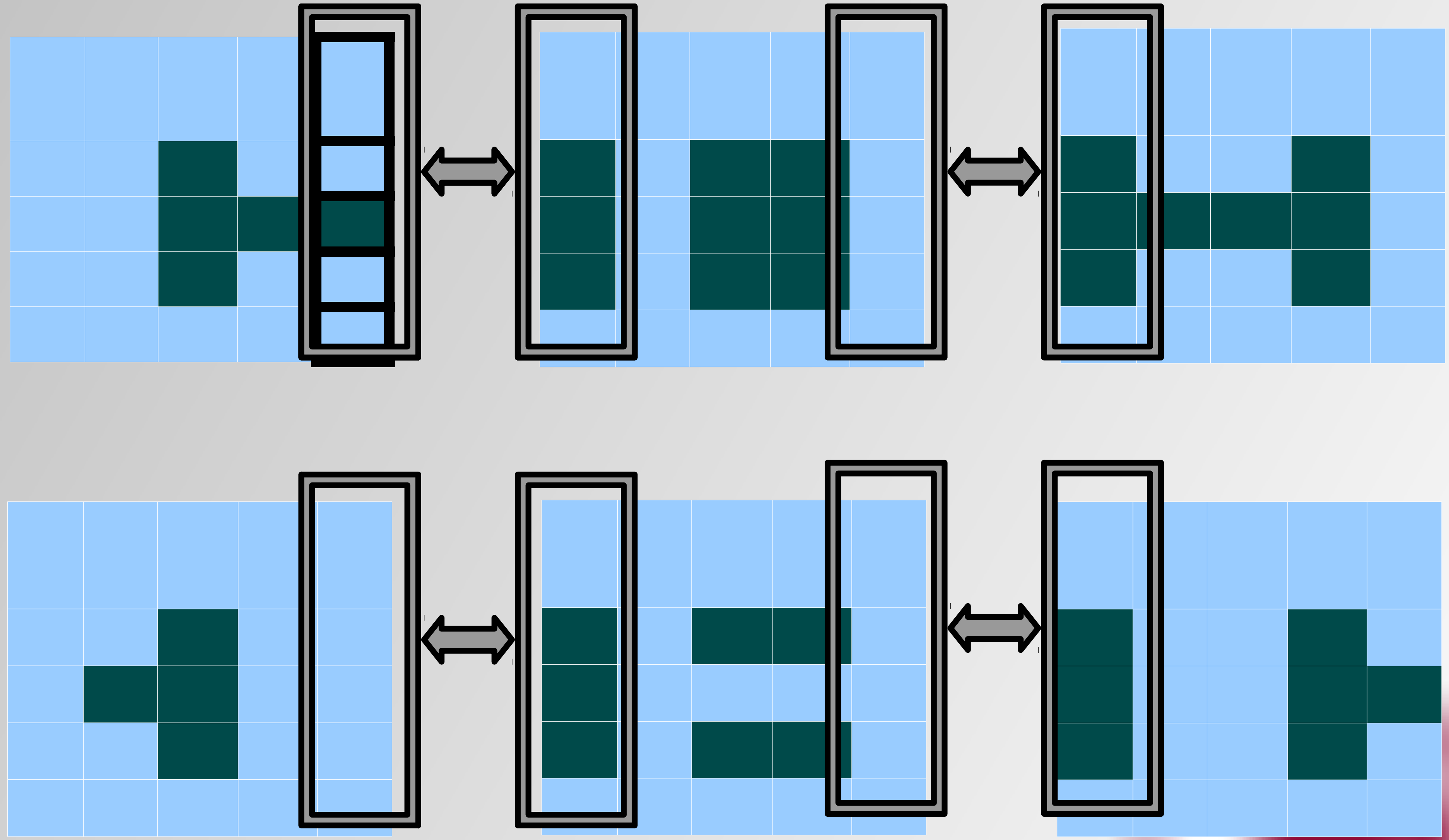
Hard to implement fault tolerance efficiently

Strategy

Divide grid into vertical slices of equal width, with one process per slice

Between each iteration, processes send and receive their extreme columns with neighbors

Example



Case study: Sample sort

- Similar to Game of Life, but even more communication intensive
- Algorithm is divided into distinct stages, allowing us to at least checkpoint between stages, if not during

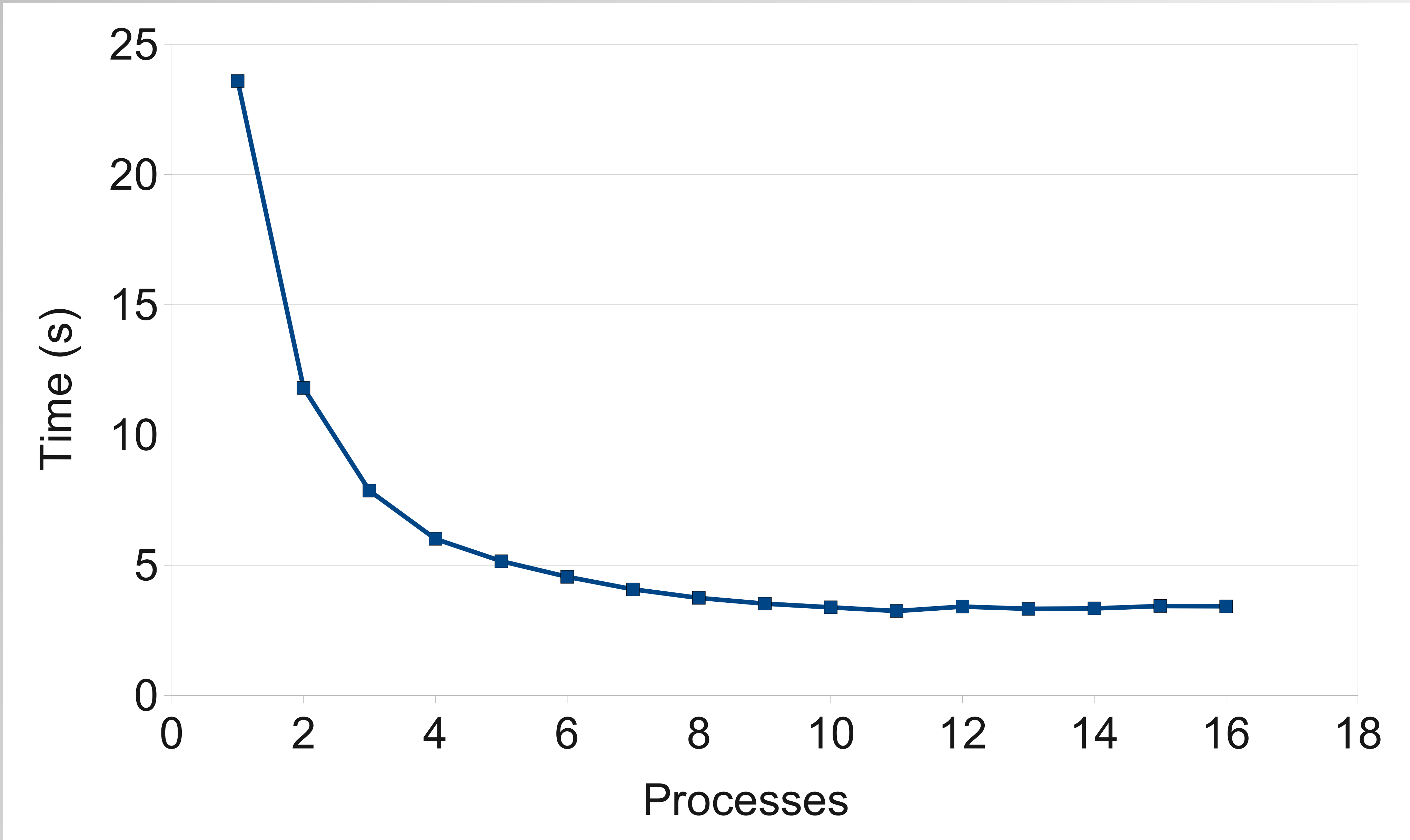
Performance

Strong and weak scaling

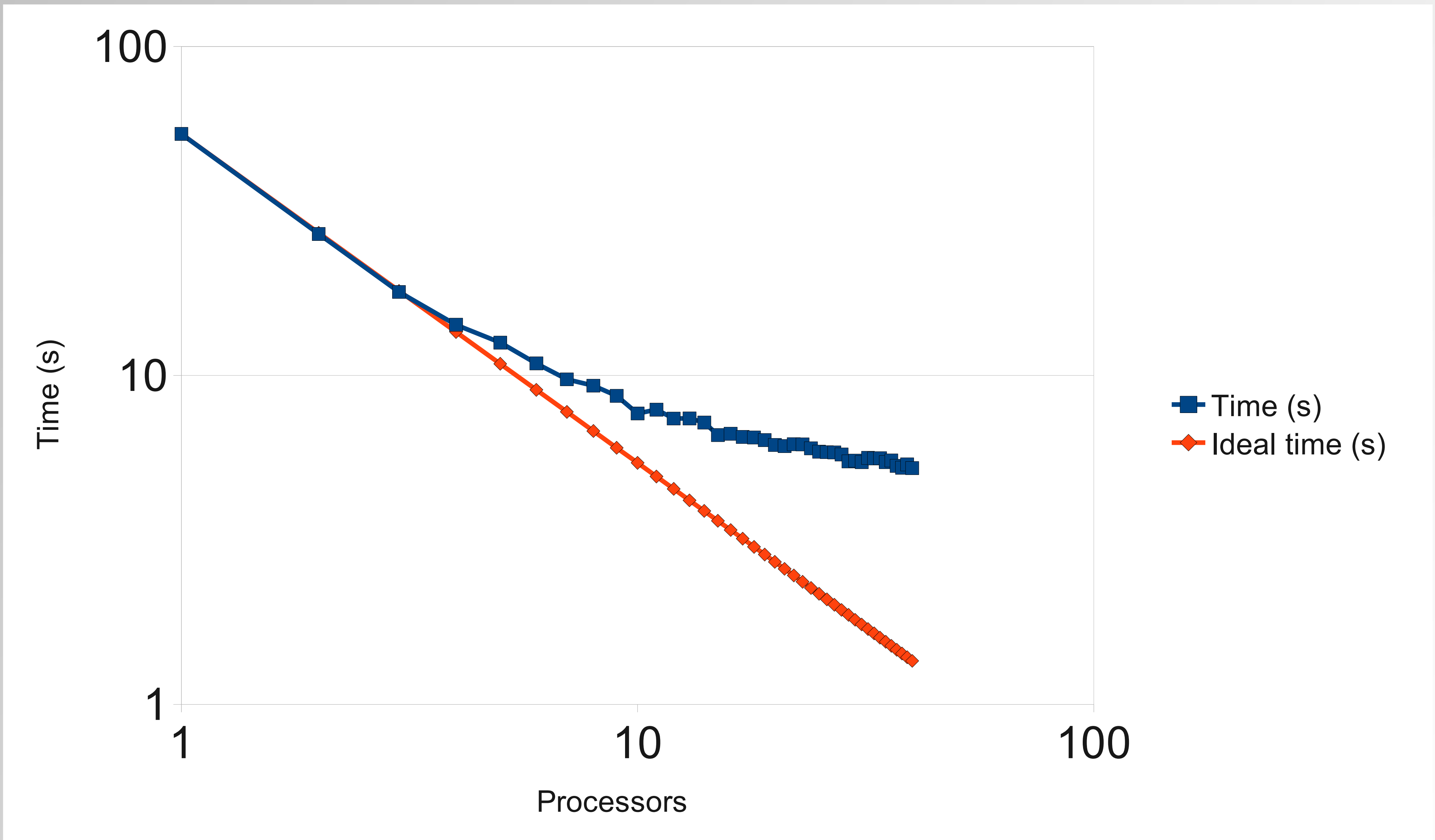
Strong scaling: Fixed problem size, variable number of processors. Expect reciprocally decreasing time.

Weak scaling: Amount of computation is made proportional to number of processors. Expect constant time.

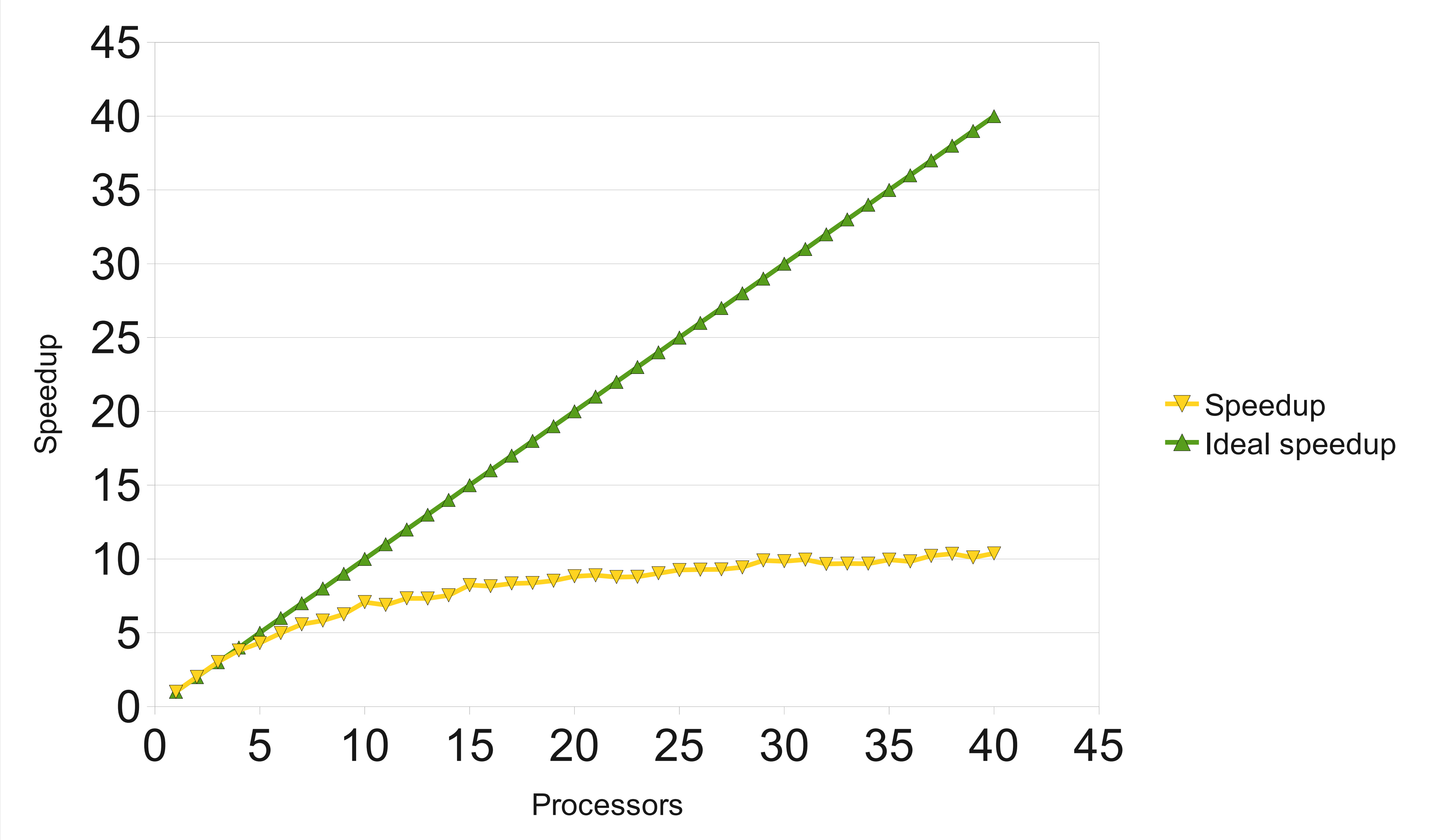
Mandelbrot, strong scaling, 1 node



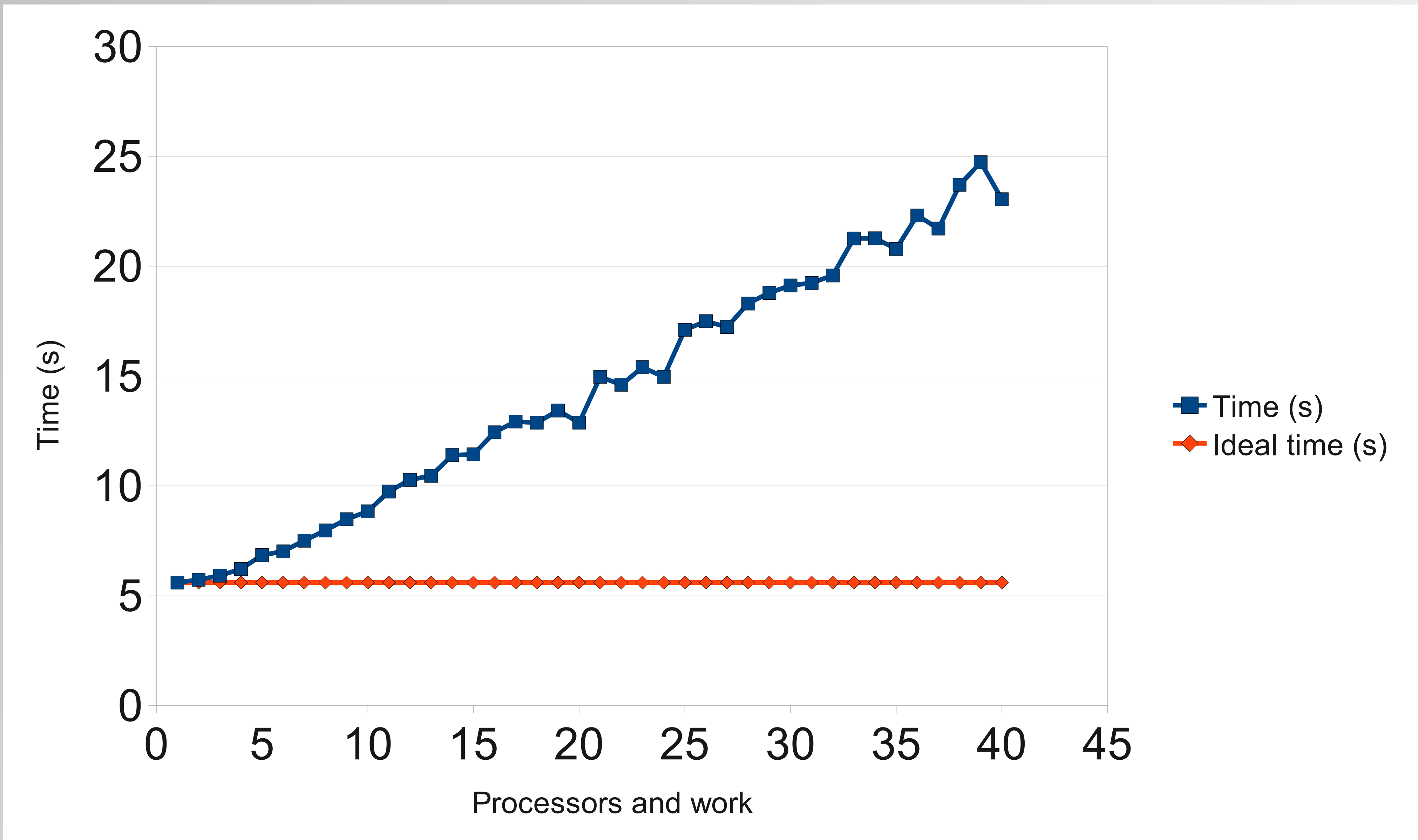
Mandelbrot, strong scaling



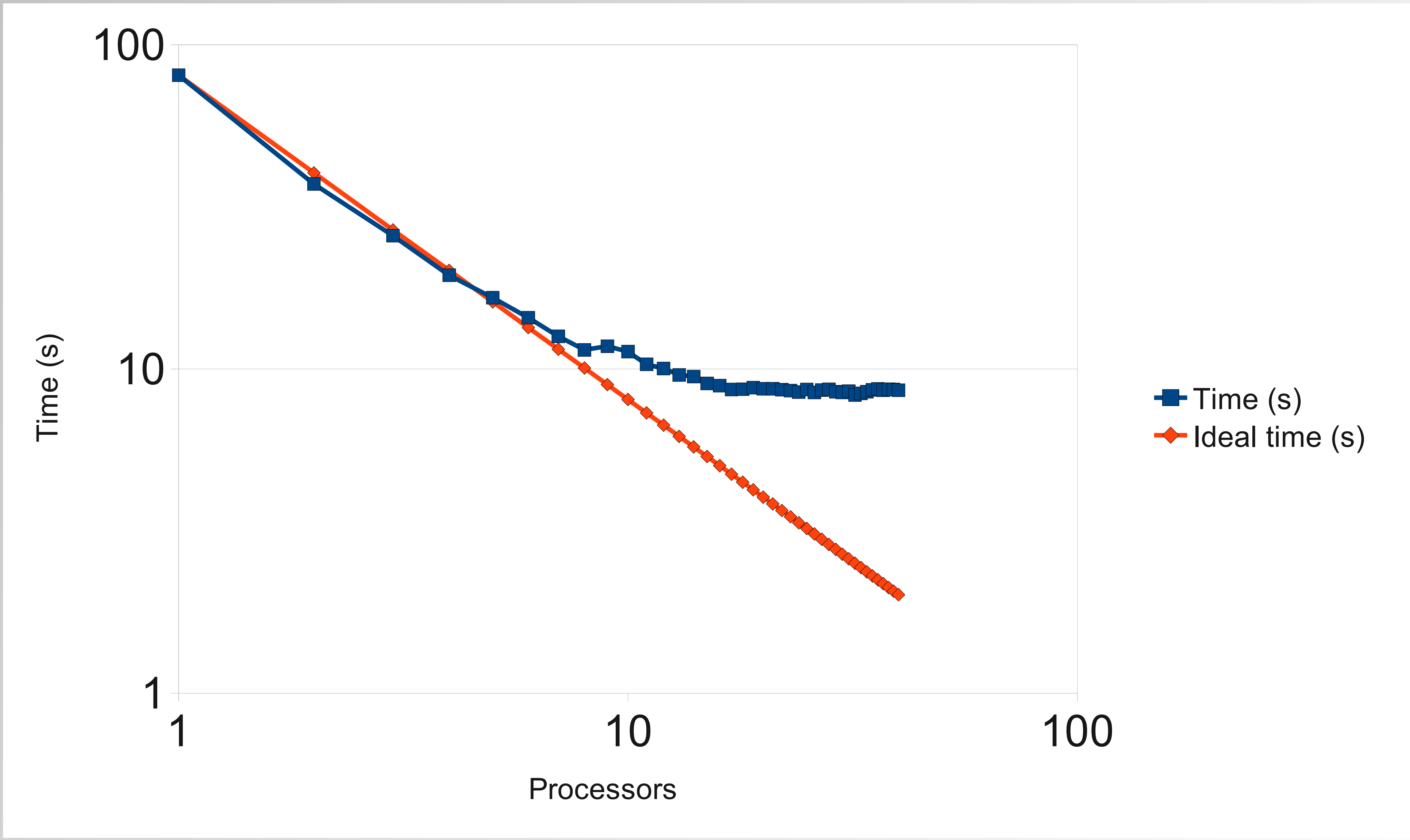
Mandelbrot, strong scaling, speedup



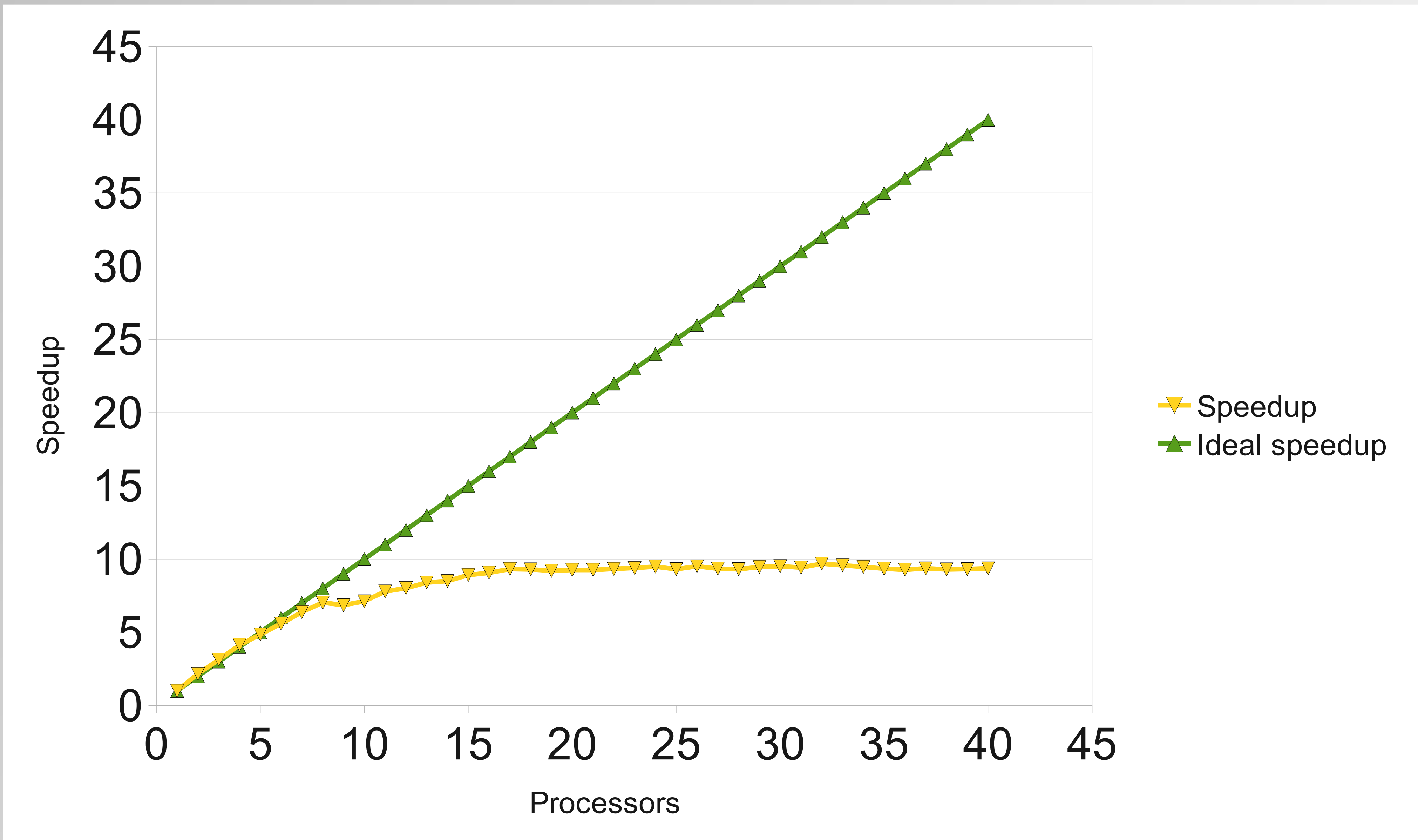
Mandelbrot, weak scaling



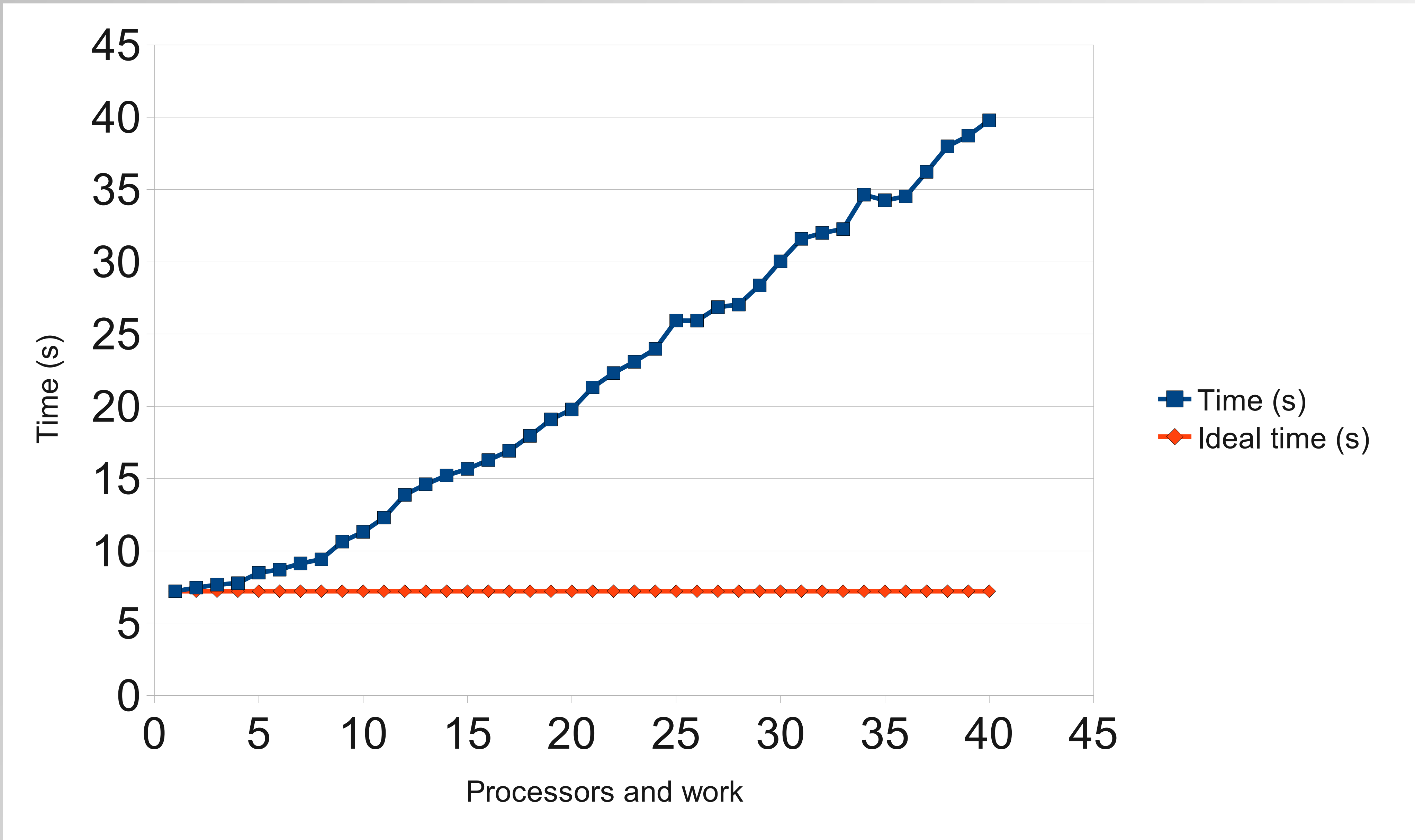
Life, strong scaling, 10 nodes



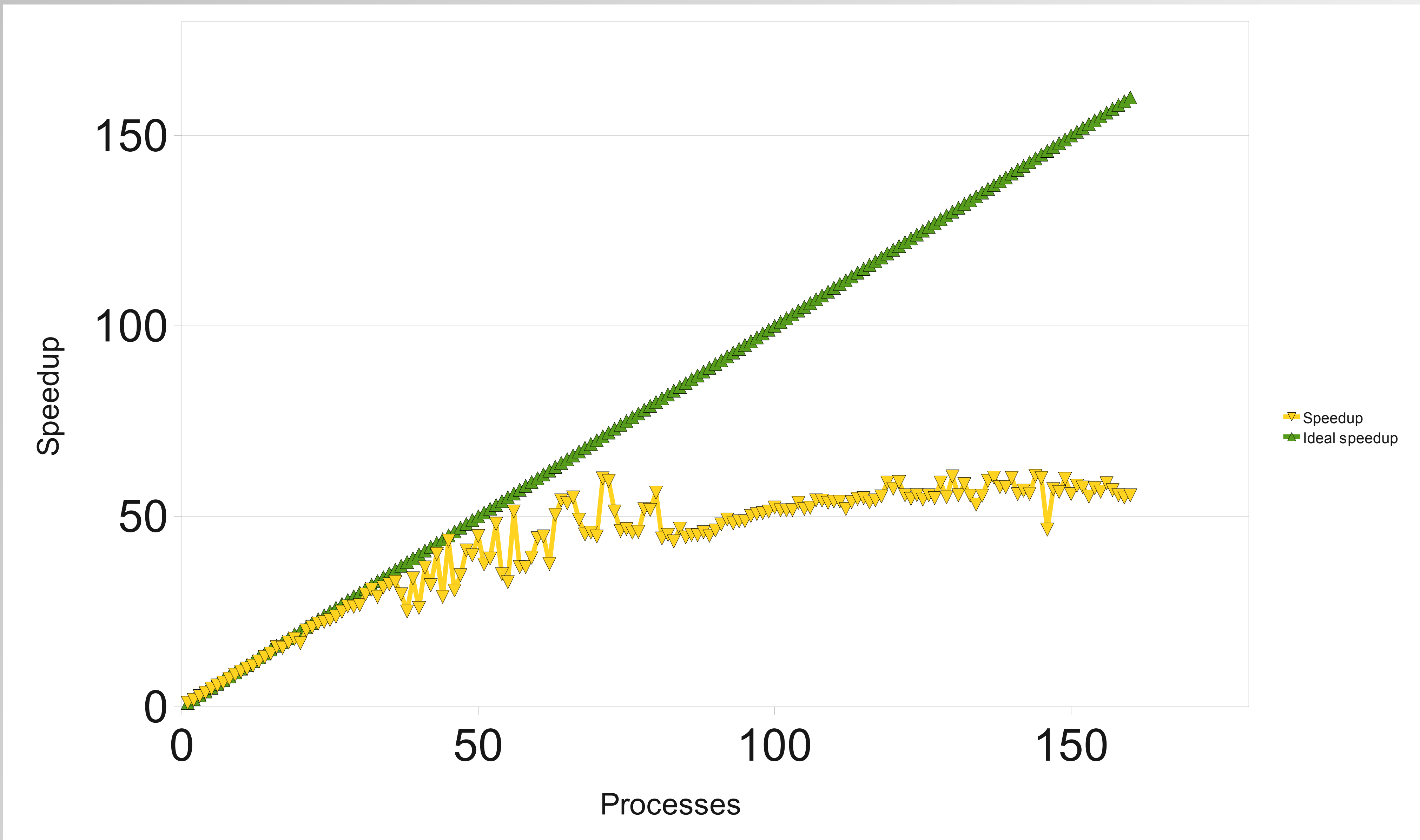
Life, strong scaling, speedup



Life, weak scaling



Simplified Mandelbrot, strong scaling



Erlang and C

Can use Erlang and C to do parallel computation

Erlang handles process control signals

C code does the heavy computation

Acknowledgements

Erlang package (open source)

Mentors: Greg Wilson, Jonathan Dursi

SciNet General Purpose Cluster (GPC)

Summary

Erlang is not a get-out-of-jail-free card

Parallelization and fault tolerance require planning, and there is no general strategy

Q&A