NEURON + Threads

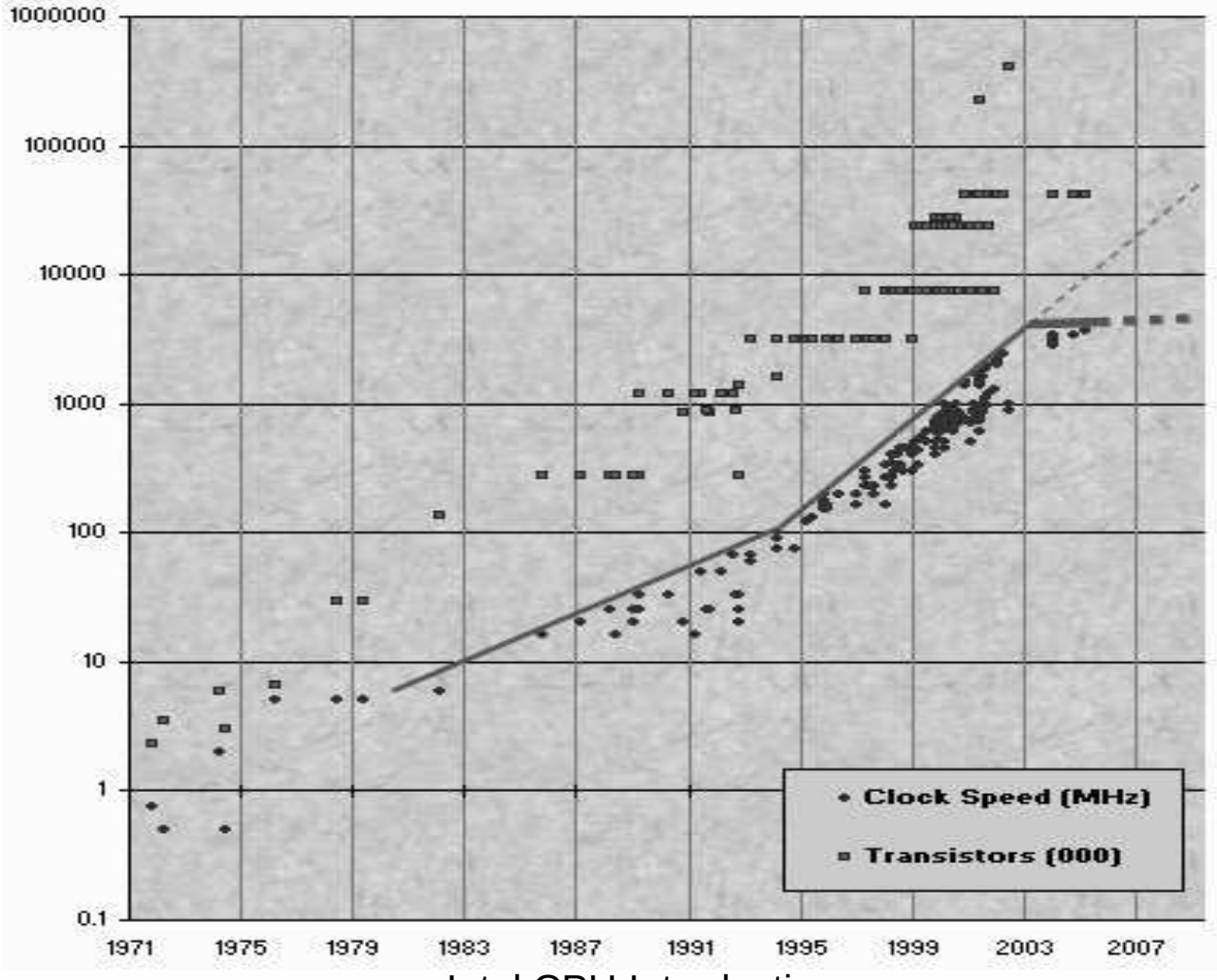
Simulations on multicore desktops.

NEURON + Threads

Simulations on multicore desktops.

Not for extracellular or LinearCircuit
Only for "large" models > 3000 states

No (more) Free Lunch



Intel CPU Introductions

(Intel, Wikipedia)

Thread style in NEURON

Join

```
 run() {
   while (t < tstop) {
     multithread_job(step)
     plot()
   }
 }

void* step(NrnThread* nt) {
    ... nt−>id ...
}
```

Thread style in NEURON

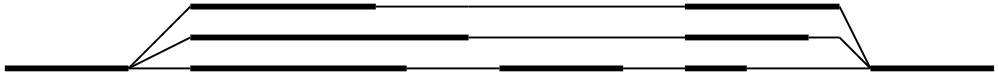
Join

```
 run() {
   while (t < tstop) {
     multithread_job(step)
     plot()
   }
 }

void* step(NrnThread* nt) {
    ... nt−>id ...
}
```

We never use.

Condition Wait

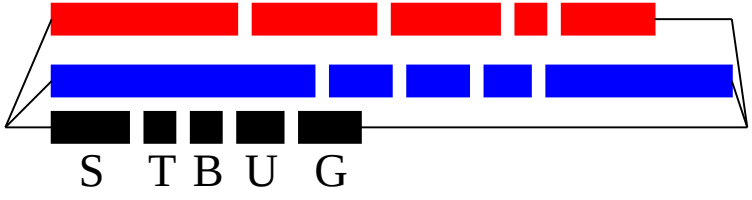
```
multithread_job(run)

run(NrnThread* nt) {
  while(t < tstop) {
    step(nt)
    barrier()
    if (nt−>id == 0) { plot() }
    barrier()
  }
}
```

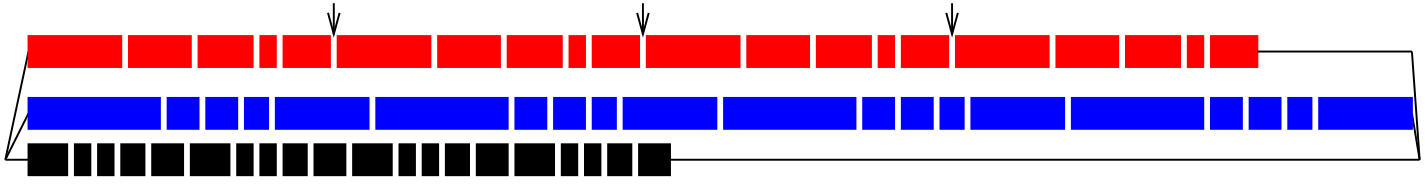


Reminiscent of MPI

Fixed step: t −> t + dt

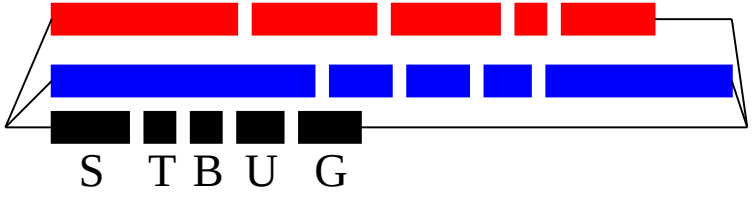
S	T	R	B	U	G
setup	triang	reduce solve	bksub	update	cond gates

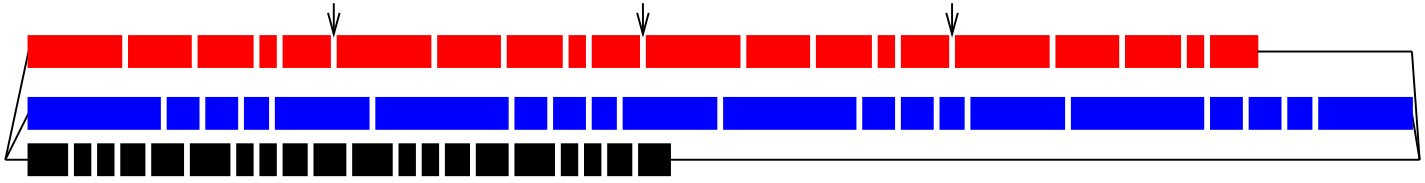


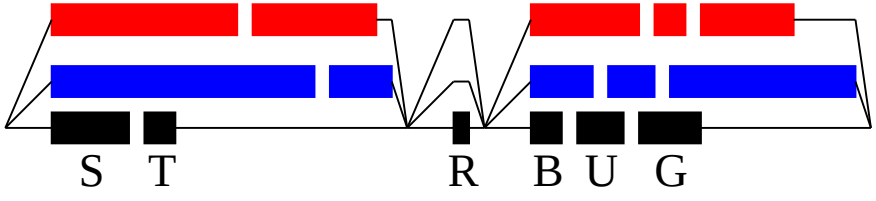


Fixed step: t −> t + dt

S	T	R	B	U	G
setup	triang	reduce solve	bksub	update	cond gates





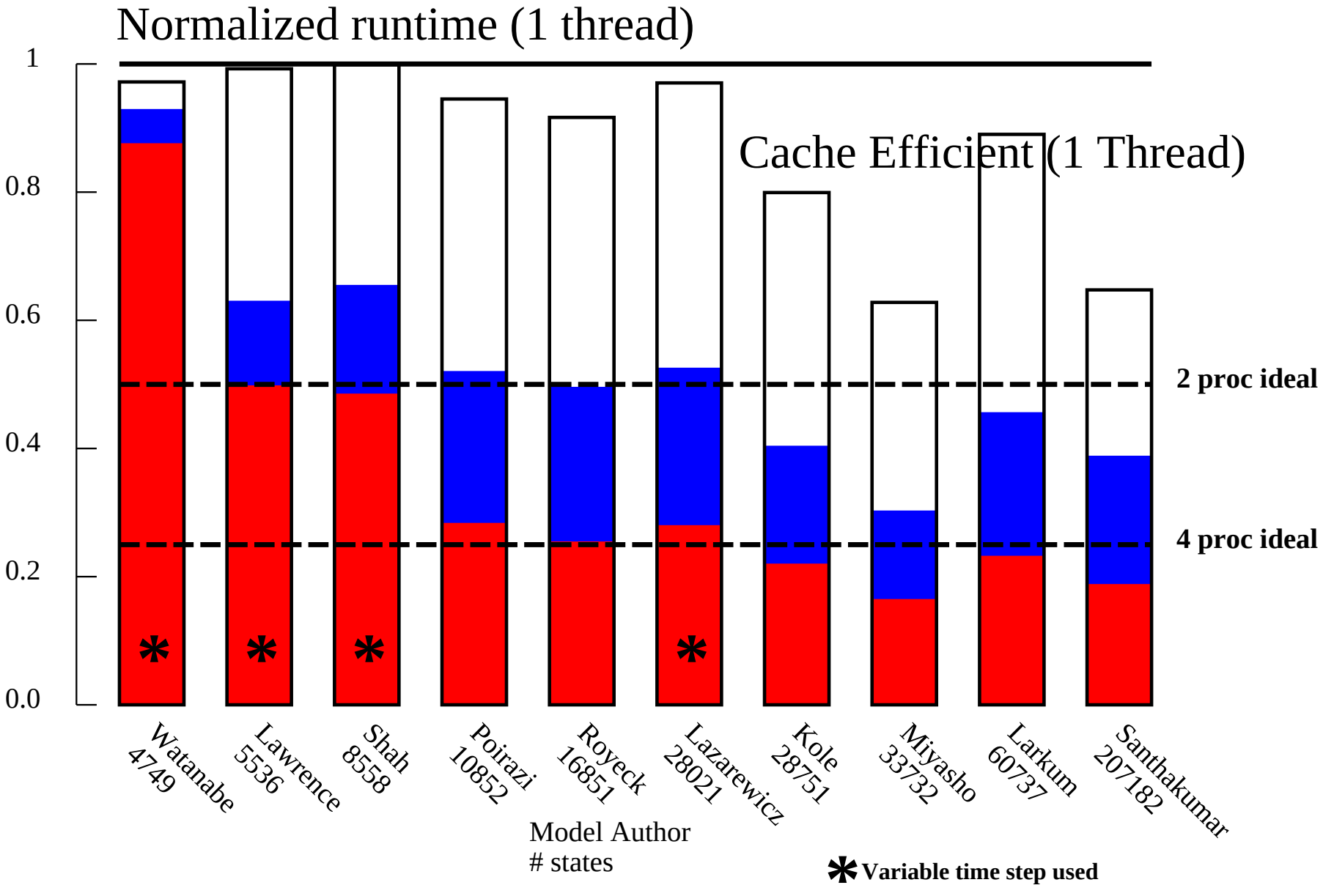


Global var dt

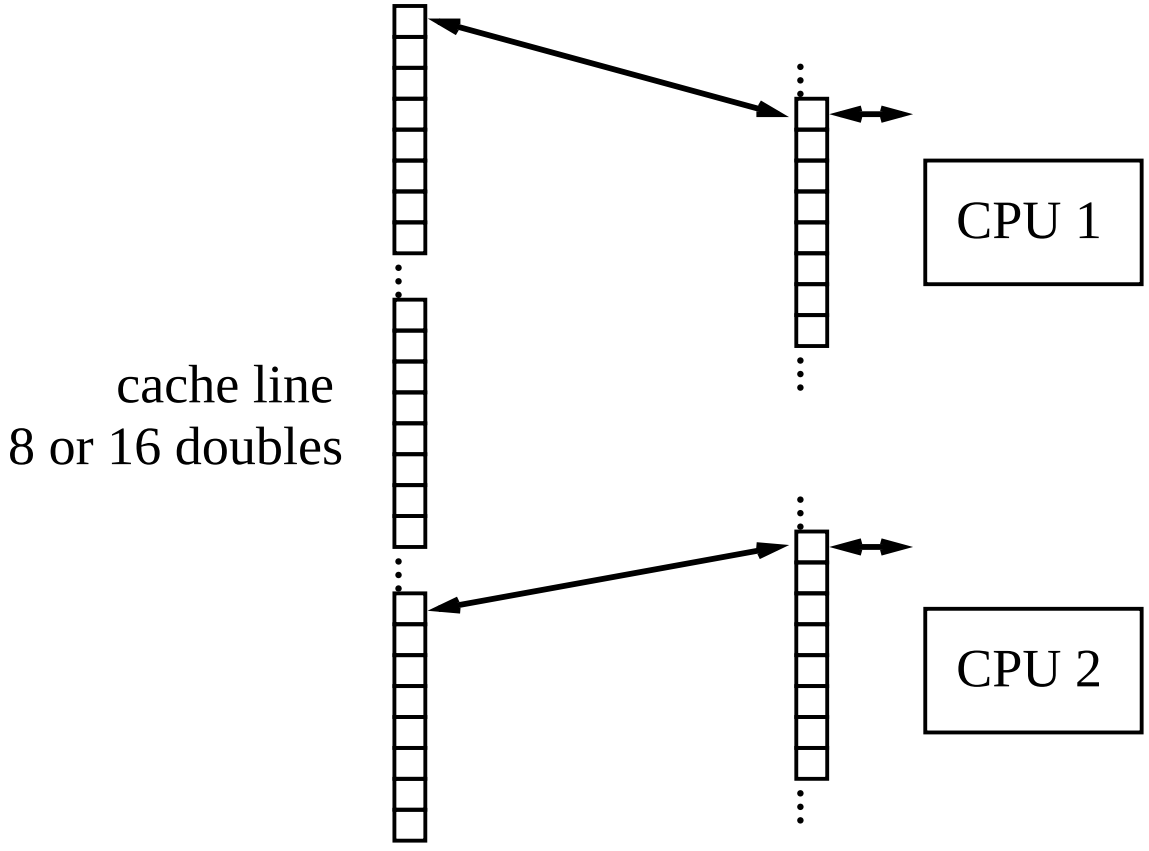
y’ = f() dy’/dy

27 ||Vector operations

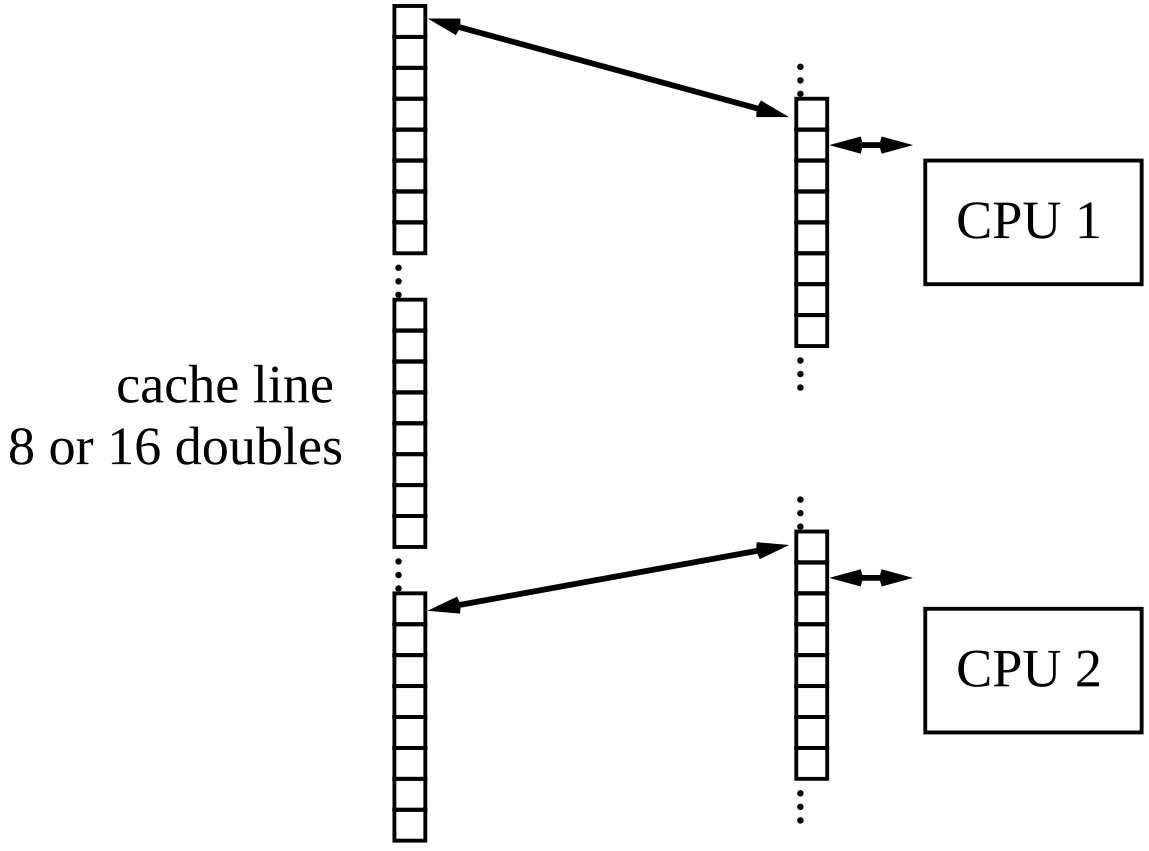
Relative thread performance
1 , 2 , 4 processors



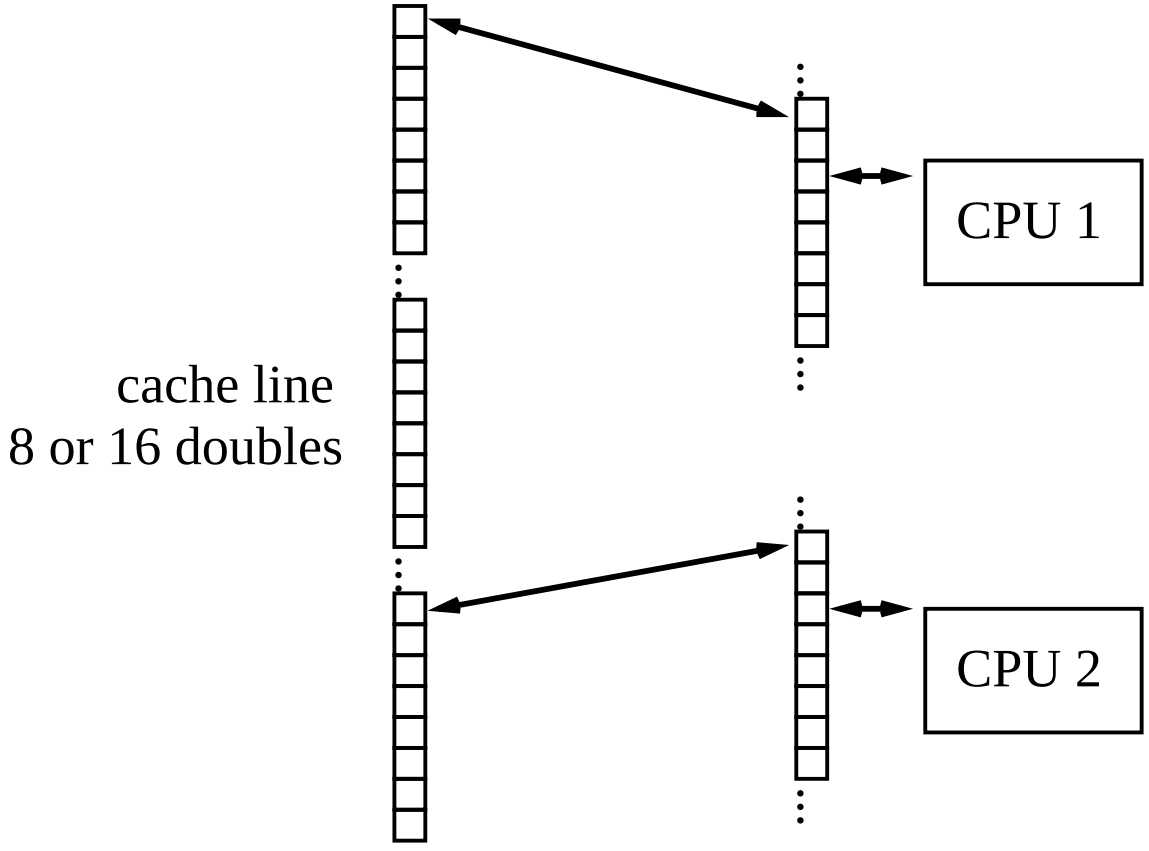
Ideal cache efficiency



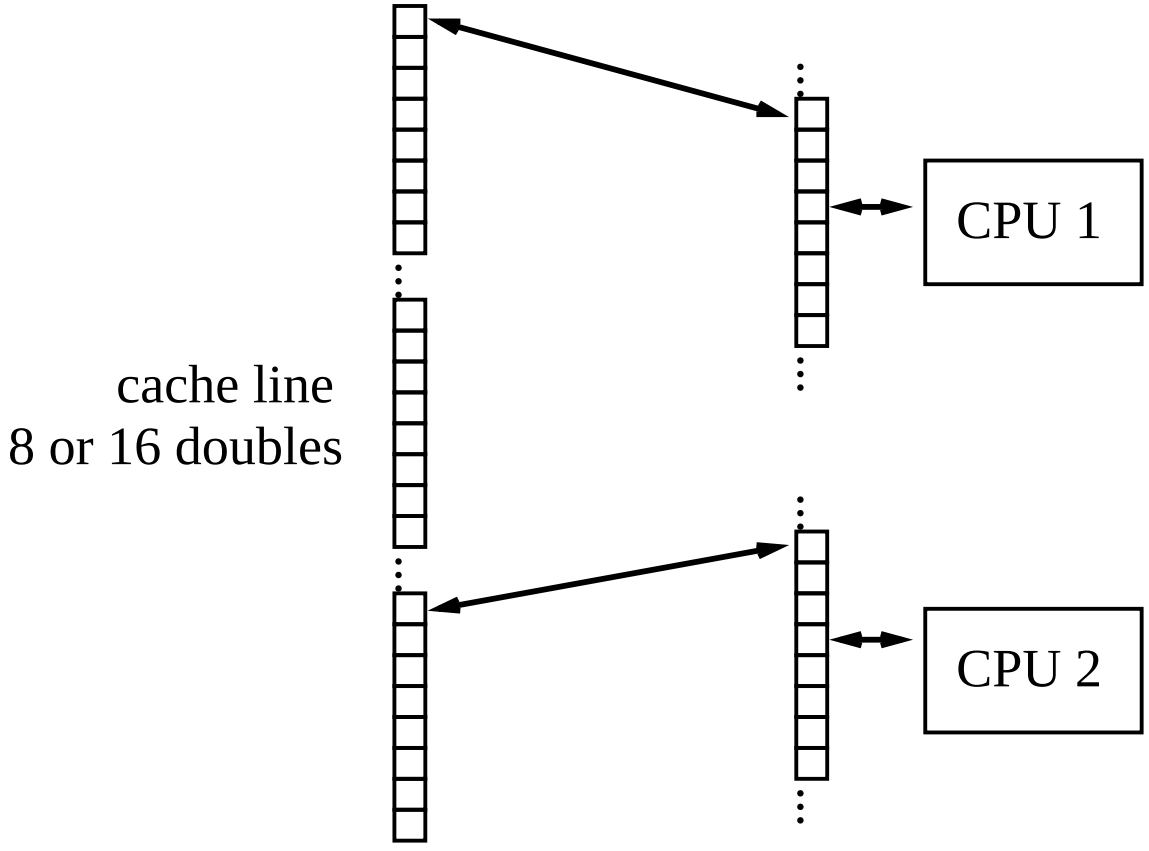
Ideal cache efficiency



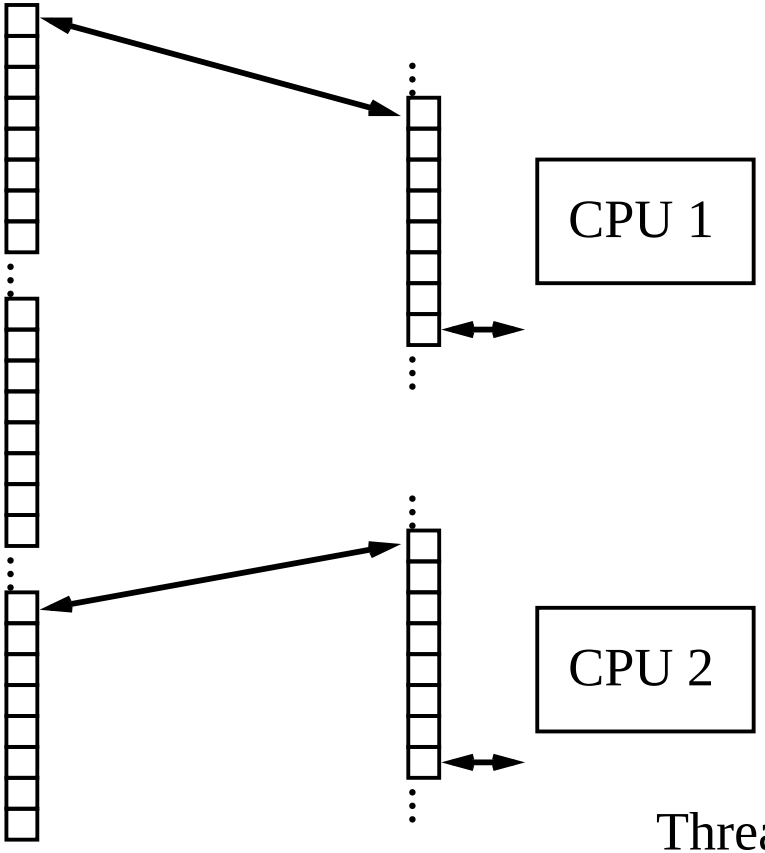
Ideal cache efficiency



Ideal cache efficiency



Ideal cache efficiency

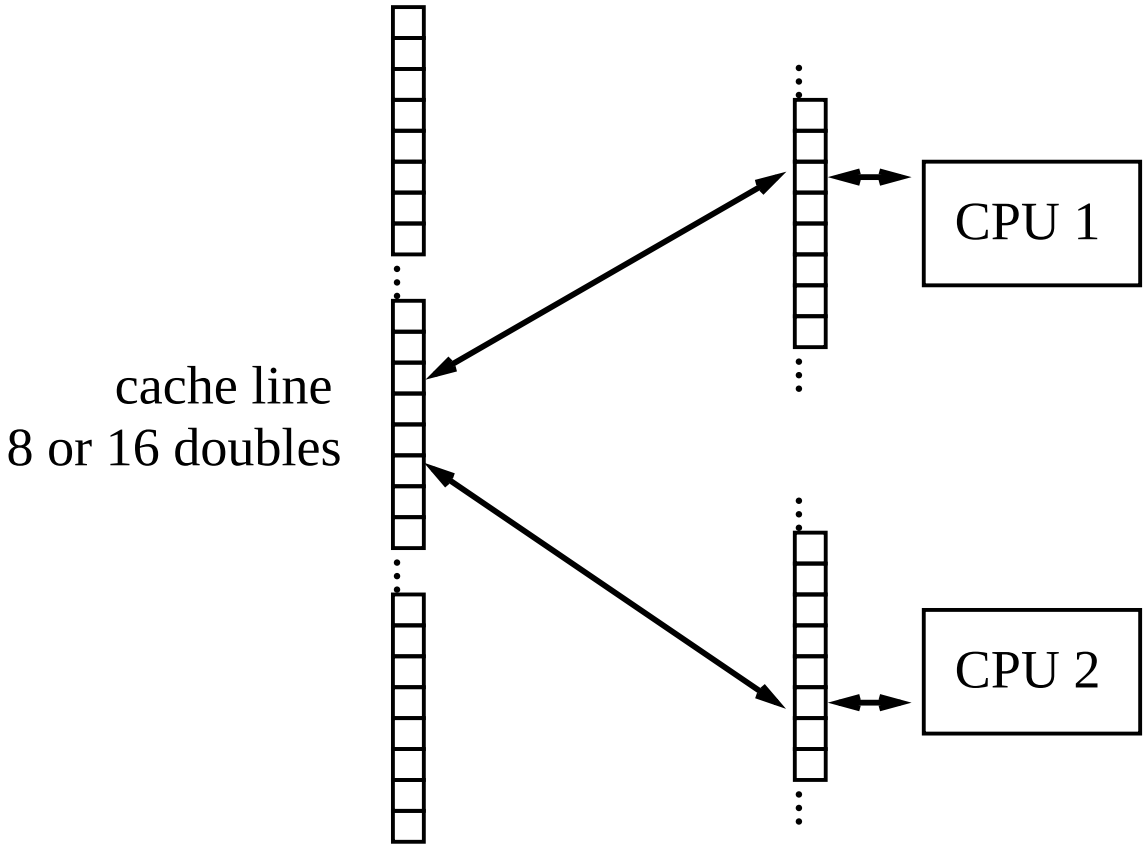


cache line
8 or 16 doubles

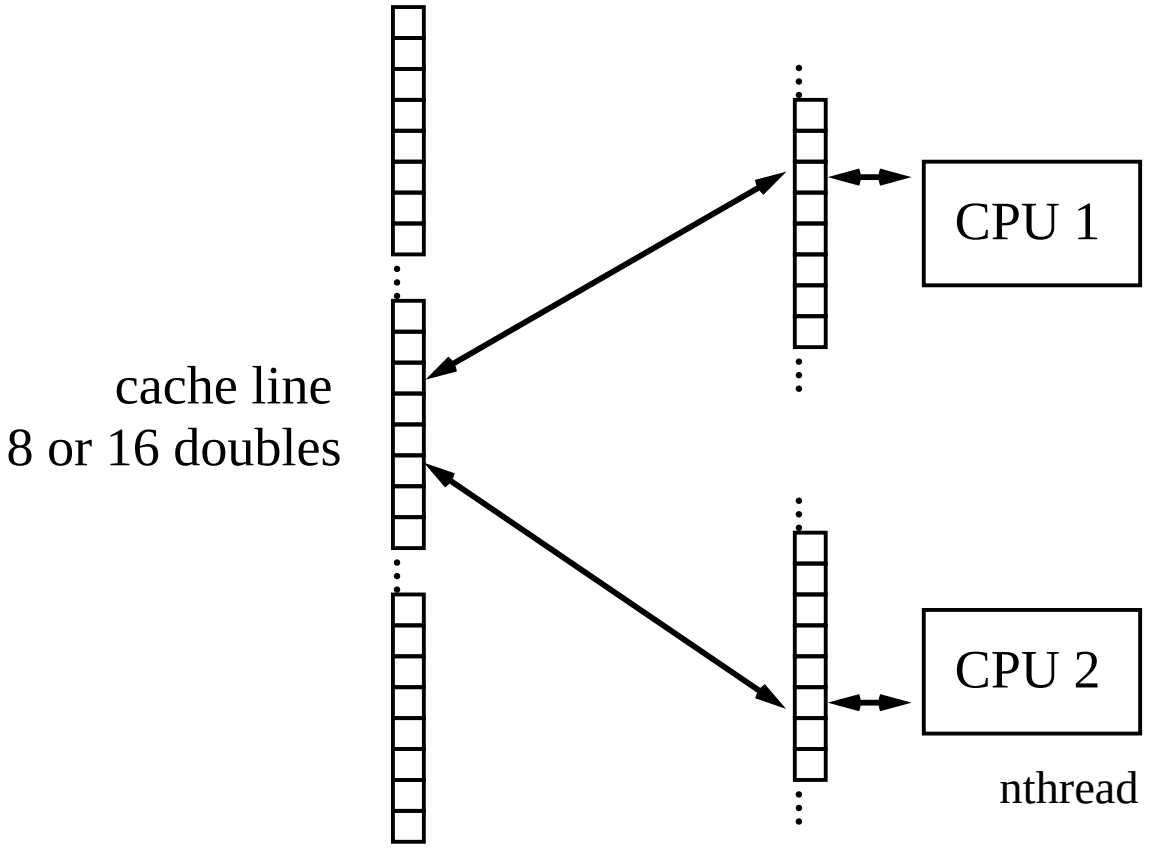
Threads	Runtime (s) Cache Efficiency Off	On
1	4.92	0.45
2	1.14	0.23
4	0.29	0.12
8	0.23	0.09

10000 passive compartments
4 core 3GHz x86_64

False cache line sharing



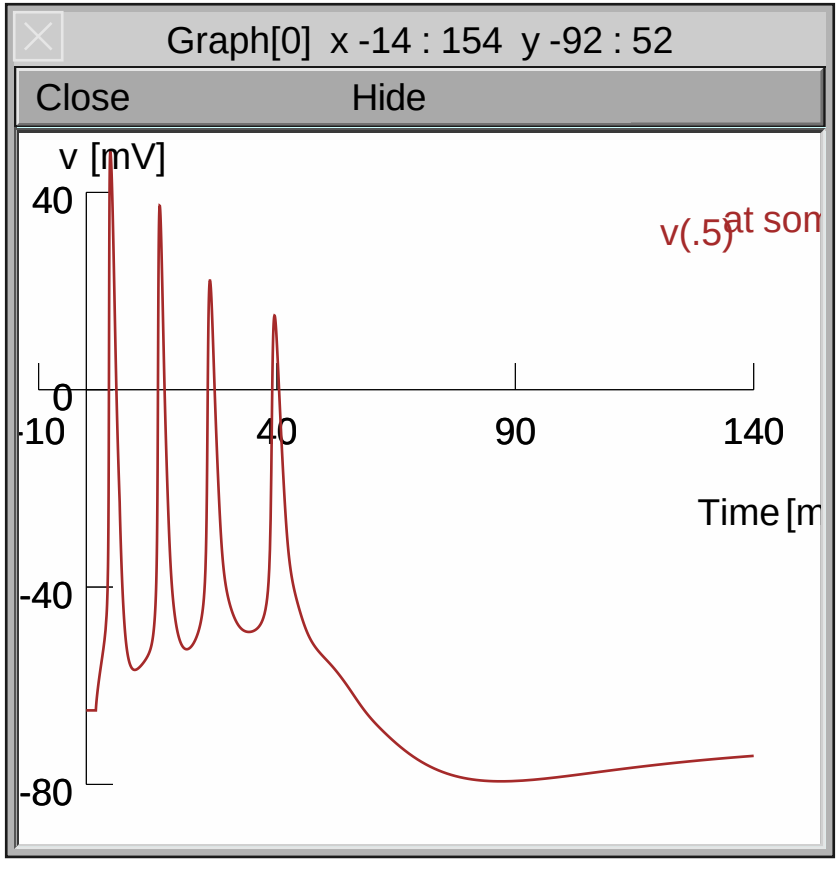
False cache line sharing

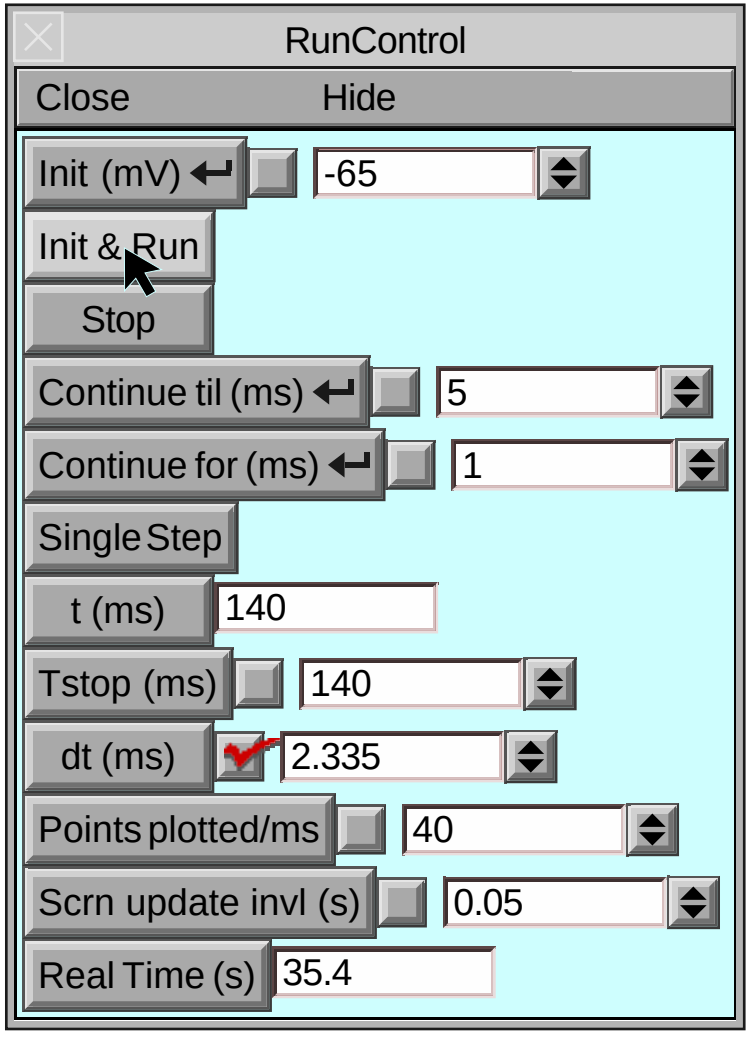


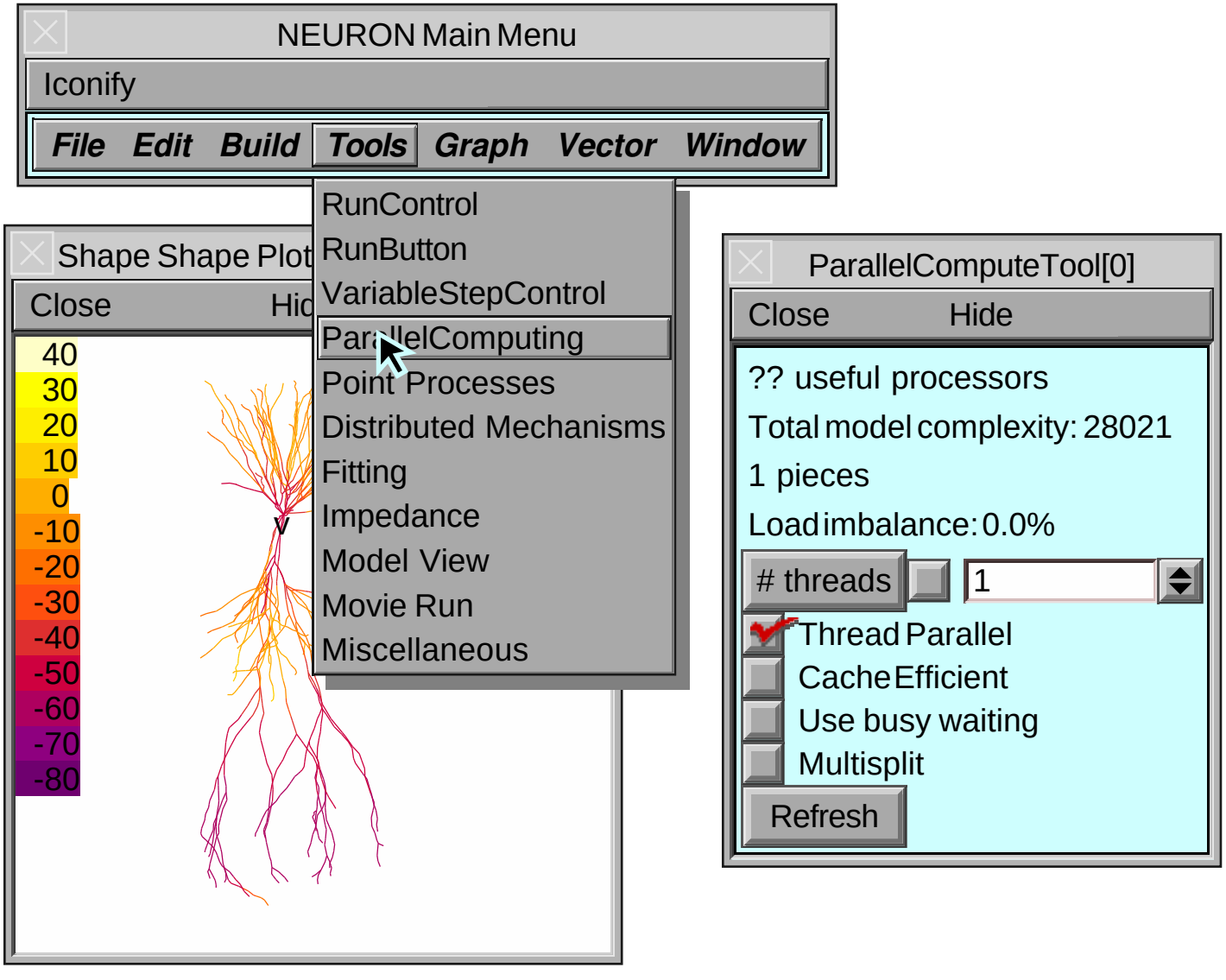

```
for ithread = 1, nthread {
  for i=1, 100million { a[ithread] += i }
}
```

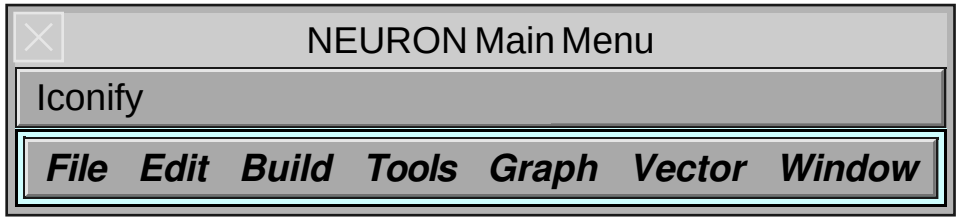
nthread	separation (bytes)	runtime (s)
1	x	0.4
4	4	4.5
4	32	5.0
4	64	0.4
8	4	15.0
8	64	0.7

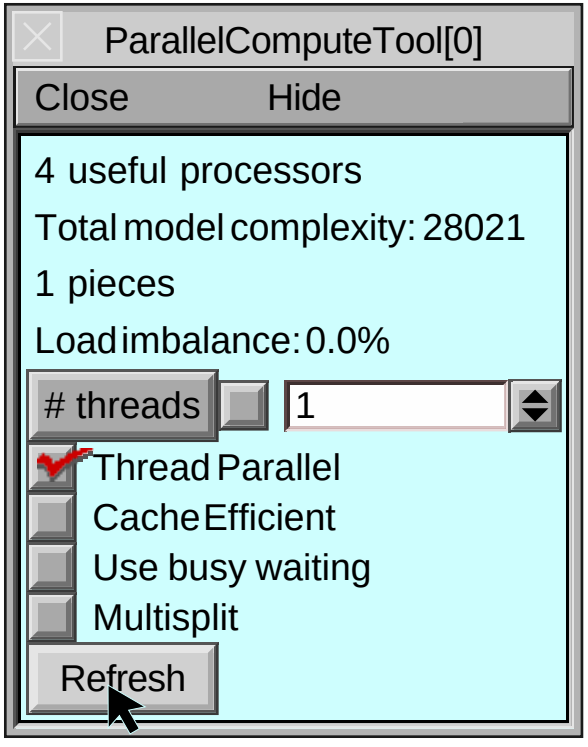
Lazarewicz 2002, CA3 Pyramidal Neuron





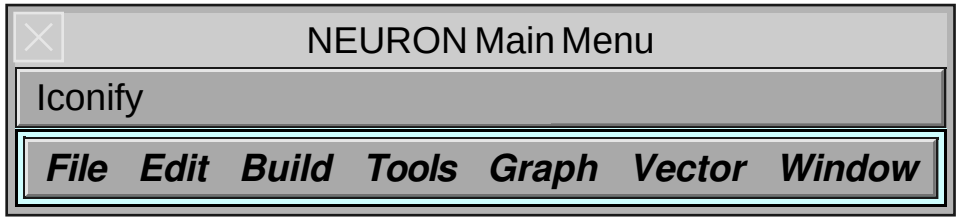

NEURON Main Menu
Iconify
File Edit Build Tools Graph Vector Window
RunControl
RunButton
VariableStepControl
ParallelComputing
Point Processes
Distributed Mechanisms
Fitting
Impedance
Model View
Movie Run
Miscellaneous
Shape Shape Plot
Close
Hid
40
30
20
10
0
-10
-20
-30
-40
-50
-60
-70
-80
v
ParallelComputeTool[0]
Close
Hide
?? useful processors
Total model complexity: 28021
1 pieces
Load imbalance: 0.0%
threads
1
Thread Parallel
Cache Efficient
Use busy waiting
Multisplit
Refresh

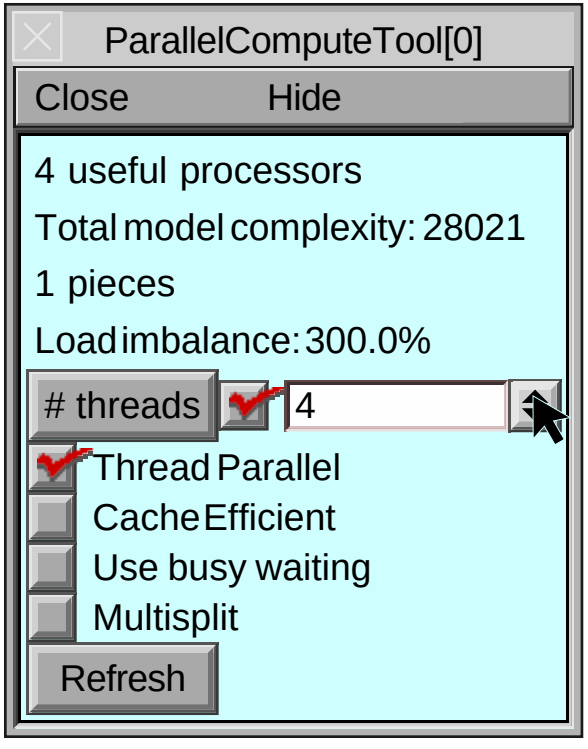




oc>nthread walltime (count to 1e8 on each thread)

1	0.0500002
2	0.0599999
4	0.0599999
8	0.14


NEURON Main Menu
Iconify
File Edit Build Tools Graph Vector Window


ParallelComputeTool[0]
Close Hide
4 useful processors
Total model complexity: 28021
1 pieces
Load imbalance: 300.0%
threads
4
Thread Parallel
Cache Efficient
Use busy waiting
Multisplit
Refresh

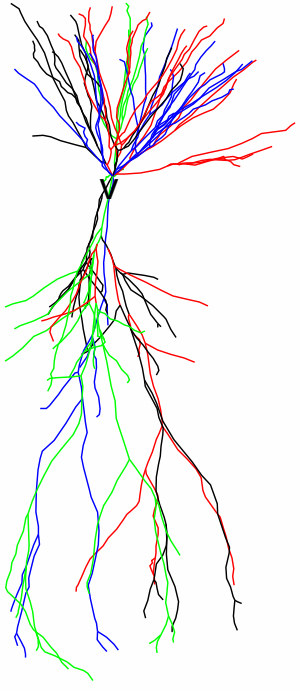
NEURON Main Menu

Iconify

File Edit Build Tools Graph Vector Window

Shape x −464.993 : 577.093 y −795.633 : 33

Close Hide



ParallelComputeTool[0]

Close Hide

4 useful processors

Total model complexity: 28044

24 pieces

Load imbalance: 1.9%

threads 4

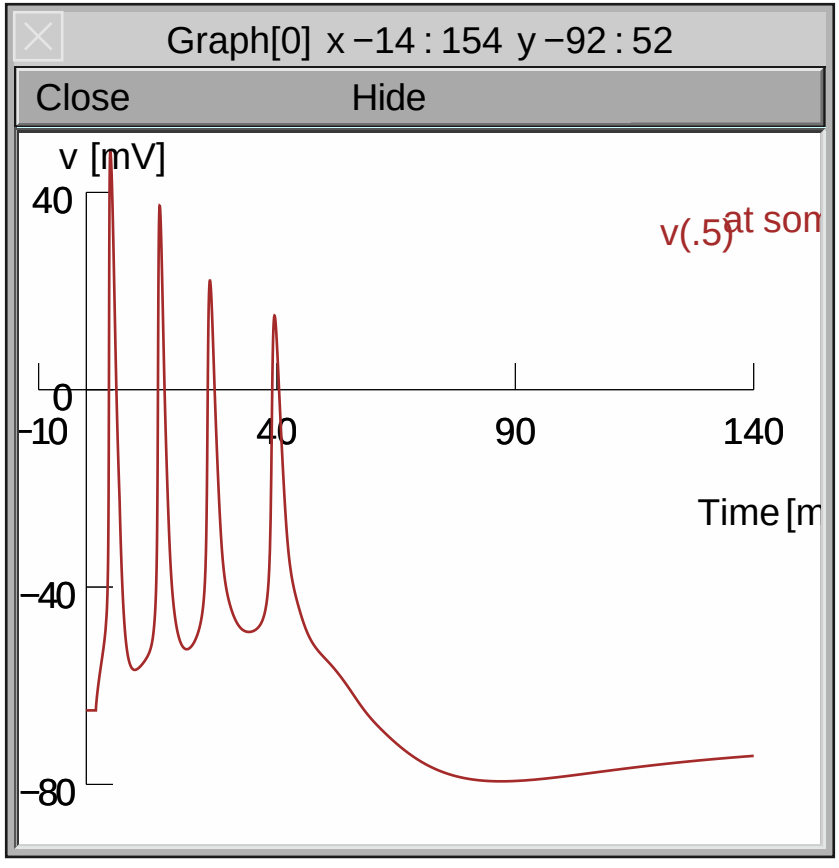
Thread Parallel

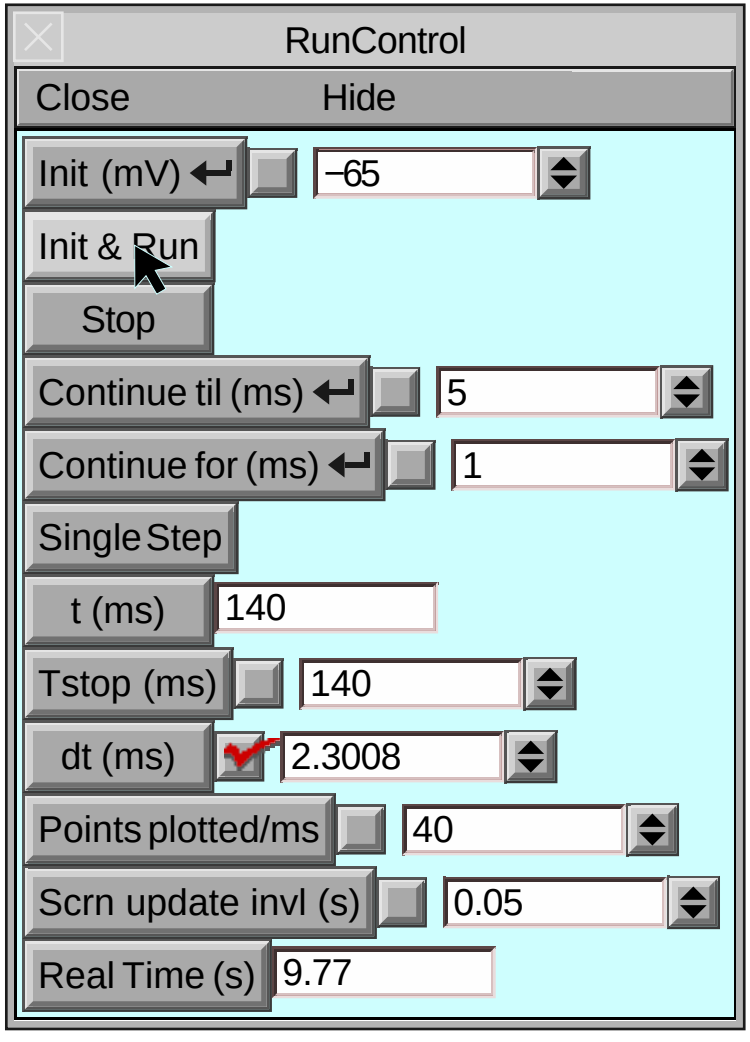
Cache Efficient

Use busy waiting

Multisplit

Refresh





instead of 35.4s

$ mkthreadsafe

```
NEURON {
        SUFFIX CAlM95
        USEION ca READ cai,cao WRITE ica
        RANGE  gbar,ica
        GLOBAL minf,tau
}
Translating CAlM95.mod into CAlM95.c
Notice: Assignment to the GLOBAL variable, "minf", is not thread safe
Notice: Assignment to the GLOBAL variable, "tau", is not thread safe
Force THREADSAFE? [y][n]: n
```

```
DERIVATIVE state {
        rate(v)
        m’ = (minf − m)/tau
}

PROCEDURE rate(v (mV)) {
        LOCAL a
        a    = alp(v)
        tau  = 1/(tfa*(a + bet(v)))
        minf = tfa*a*tau
}
```

Force THREADSAFE? [y][n]: n
y

```
NEURON {
    THREADSAFE
        SUFFIX CAlM95
        USEION ca READ cai,cao WRITE ica
        RANGE  gbar,ica
        GLOBAL minf,tau
}
```

$ mkthreadsafe

```
NEURON {
        POINT_PROCESS GABAa
        POINTER pre
        ...
}
        VERBATIM
        return 0;
        ENDVERBATIM
Translating gabaa.mod into gabaa.c
Notice: Use of POINTER is not thread safe.
Notice: VERBATIM blocks are not thread safe
Notice: Assignment to the GLOBAL variable, "Rtau", is not thread safe
Notice: Assignment to the GLOBAL variable, "Rinf", is not thread safe
Force THREADSAFE? [y][n]: n
```

$ mkthreadsafe

```
NEURON {
    SUFFIX Kv
    USEION k READ ek WRITE ik
    RANGE n, gk, gbar
    RANGE ninf, ntau
    GLOBAL Ra, Rb
    GLOBAL q10, temp, tadj, vmin, vmax
}
Translating kv.mod into kv.c
Notice: This mechanism cannot be used with CVODE
Notice: Assignment to the GLOBAL variable, "tadj", is not thread safe
Force THREADSAFE? [y][n]: n
```

```
NEURON {
    GLOBAL q10, temp, tadj, vmin, vmax

INITIAL {
    trates(v)
    n = ninf
}

BREAKPOINT {
    SOLVE states
    gk = tadj*gbar*n
    ik = (1e−4) * gk * (v − ek)
}

PROCEDURE trates(v) {
    TABLE ninf, nexp
    tadj = q10^((celsius − temp)/10)
```

```
NEURON {        THREADSAFE
    GLOBAL q10, temp, tadj, vmin, vmax

INITIAL {
    trates(v)
    n = ninf
}

BREAKPOINT {
    SOLVE states
    gk = tadj*gbar*n
    ik = (1e−4) * gk * (v − ek)
}

PROCEDURE trates(v) {
    TABLE ninf, nexp
    tadj = q10^((celsius − temp)/10)
```

tadj = q10^((celsius − temp)/10)

... a case often seen in ca accumulation models

```
NEURON {
        GLOBAL vol, Buffer0


...
INITIAL {

    if (coord_done == 0) {
        coord_done = 1
        coord()
    }


...
    vol[0] = 0
    FROM i=0 TO NANN−2 {
        vol[i] = vol[i] + PI*(r−dr2/2)*2*dr2
...
        vol[i+1] = PI*(r+dr2/2)*2*dr2
```

```
NEURON {
        GLOBAL vol, Buffer0
        THREADSAFE vol
...
INITIAL {
    MUTEXLOCK
    if (coord_done == 0) {
        coord_done = 1
        coord()
    }
    MUTEXUNLOCK
...
    vol[0] = 0
    FROM i=0 TO NANN−2 {
        vol[i] = vol[i] + PI*(r−dr2/2)*2*dr2
...
        vol[i+1] = PI*(r+dr2/2)*2*dr2
```

If thread results differ,
a good way to diagnose the
cause is to use prcellstate.hoc

If thread results differ,
a good way to diagnose the
cause is to use prcellstate.hoc

$ nrngui mosinit.hoc ../prcellstate.hoc

```
// serial model
finitialize(−70)
prtop(0) // constructs cs0.0.1 (5MB)

//switch to 4 threads
finitialize(−70)
prtop(1) // constructs cs1.0.1
```

diff cs*|more

notice differences in ik and ica

and in particular

595,605c595,605
< 0 594 0.29053584721744774 gk_km(0.0454545)
< 0 595 0.29053584721744774 gk_km(0.136364)
−−−
> 0 594 0 gk_km(0.0454545)
> 0 595 0 gk_km(0.136364)

672,682c672,682
< 0 671 7.832147840514193e−12 gca_sca(0.0454545)
< 0 672 7.832147840514193e−12 gca_sca(0.136364)
−−−
> 0 671 0 gca_sca(0.0454545)
> 0 672 0 gca_sca(0.136364)