



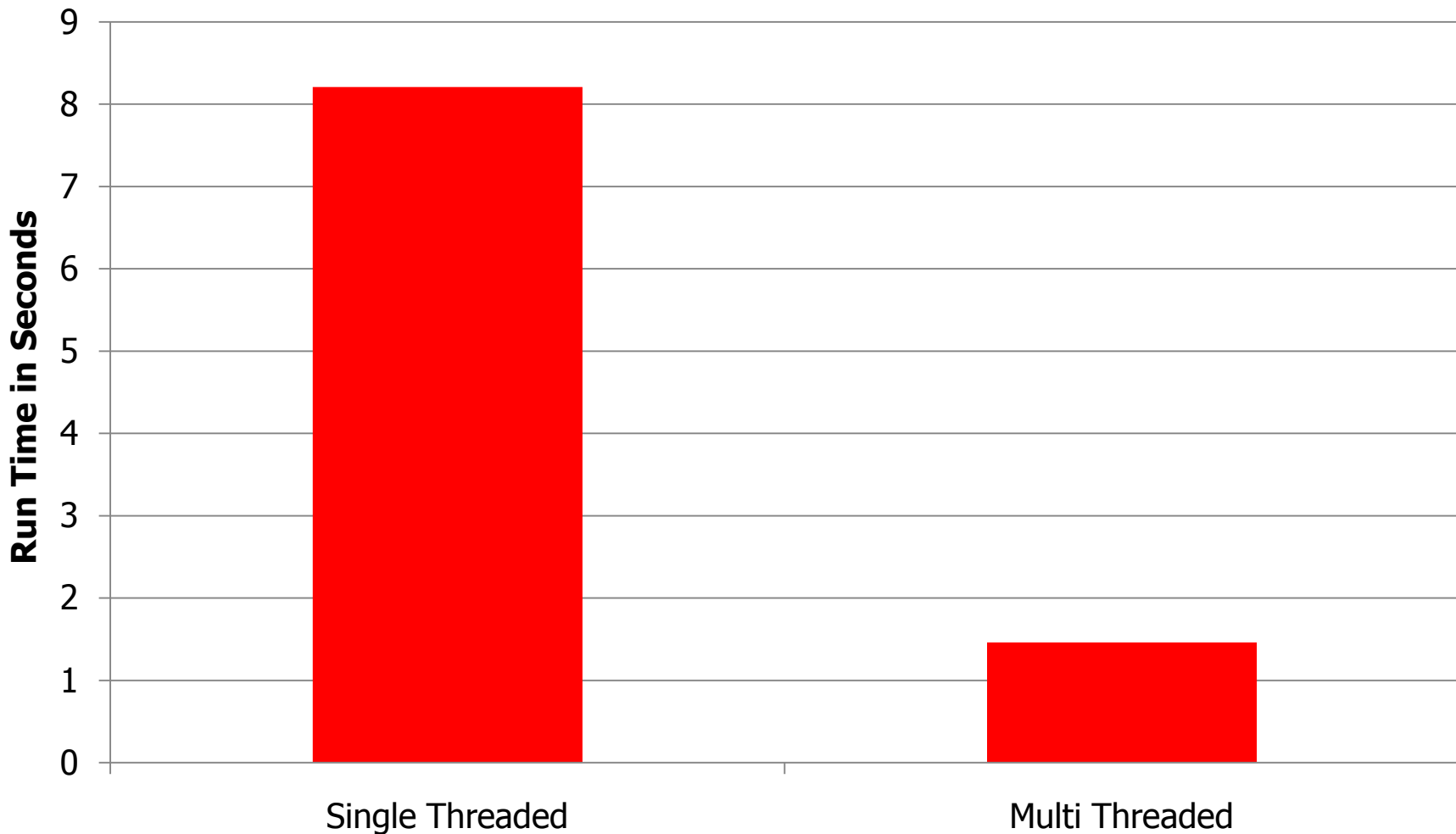
# OEChem Meets OpenMP

Brian (you're-better-off-calling-me-Bruce) Cole

CUP X Toolkits Session

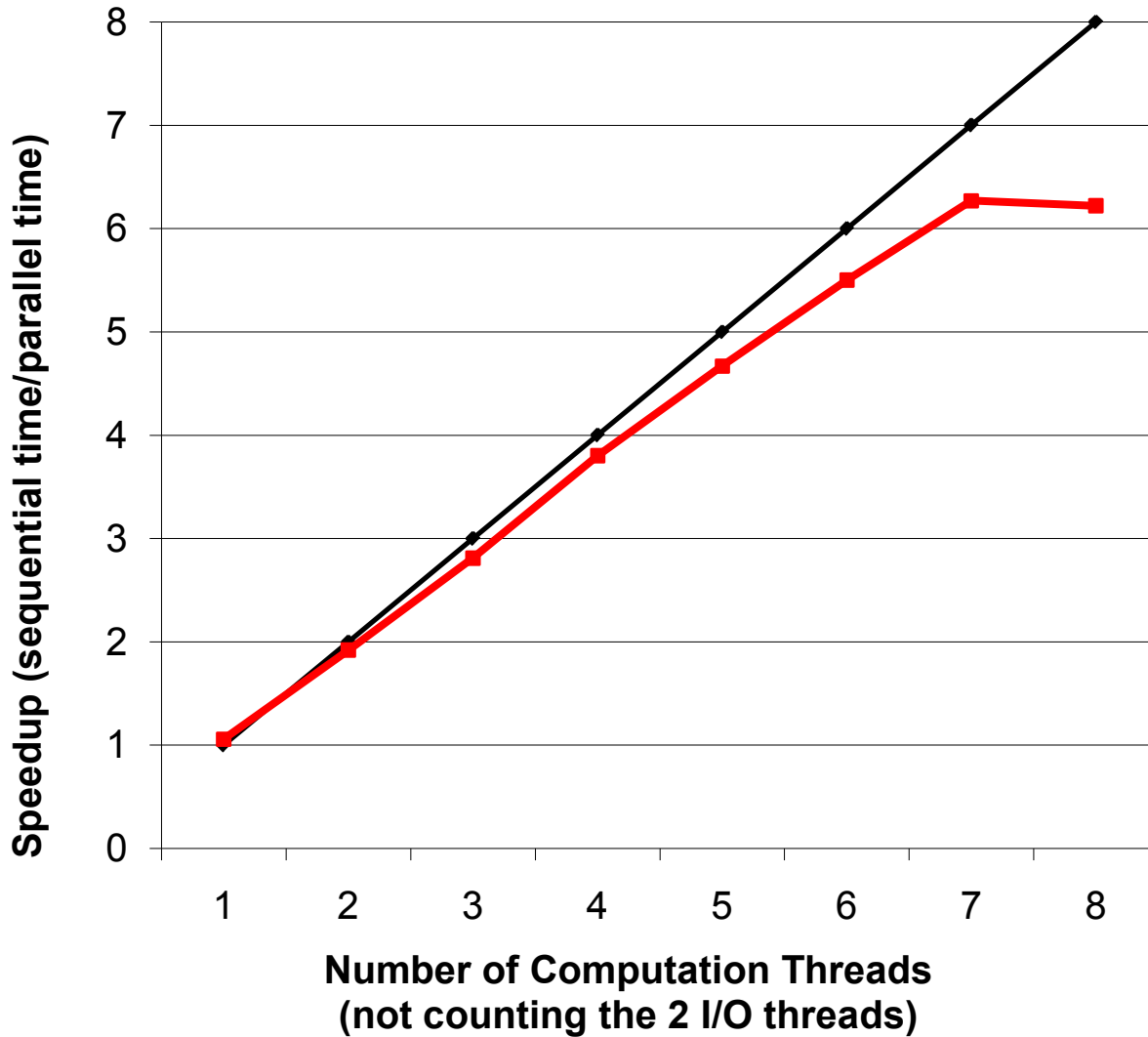


# A convincing bar graph





# Threaded molgrep speedup



**Ideal Linear Scaling**

**Actual Speedup**



# OpenMP is not OpenMPI

## OpenMP

- Threads
- Shared memory
  - Multi-core single machine
- No end-user interaction
  - Automatic scaling
- Cheap communication through synchronization
  - ~nanoseconds

## OpenMPI

- Processes on nodes
- Distributed memory
  - Cluster
- End-user interaction
  - mpirun
- Expensive communication over the network
  - ~milliseconds



# OpenMP: An API for Multithreaded Applications

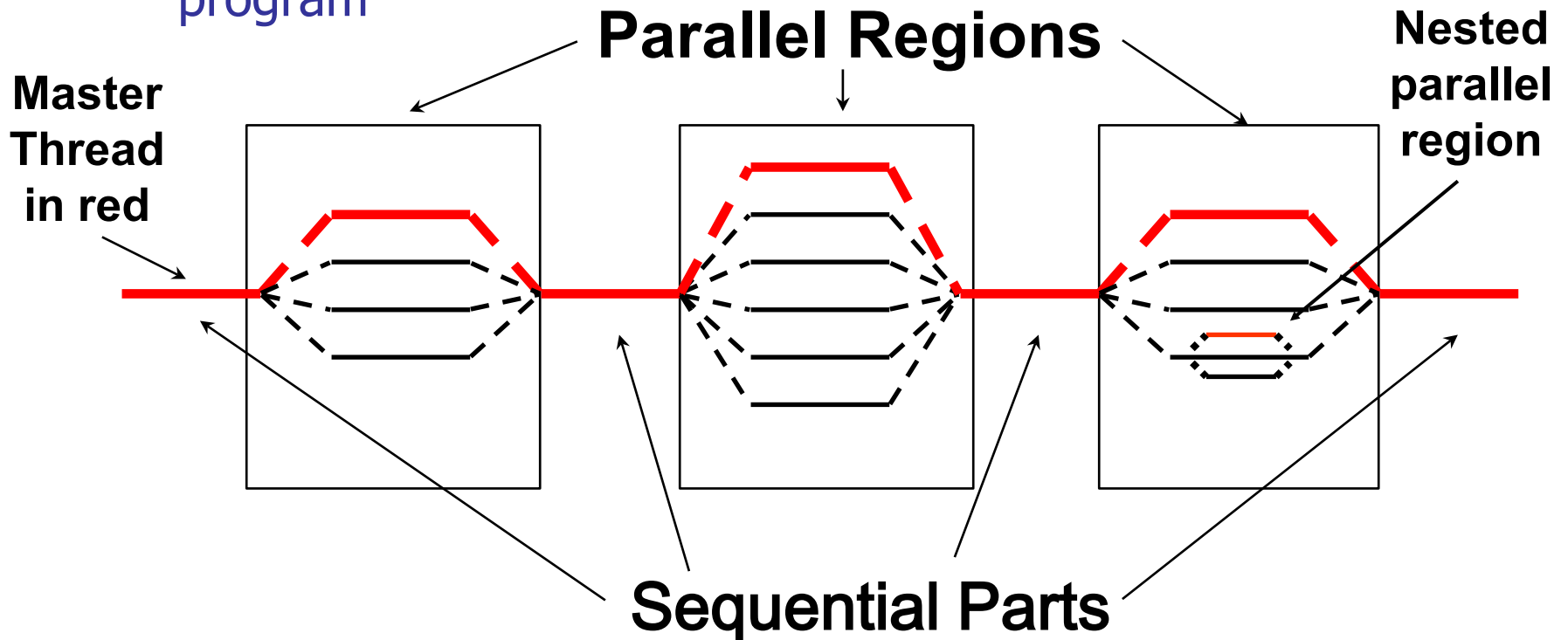
- A set of compiler directives and library routines for parallel application programmers
- Greatly simplifies writing multi-threaded (MT) programs in Fortran, C and C++
- Standardizes last 20 years of SMP practice
- Cross platform
  - GCC  $\geq$  4.2
  - MSVC  $\geq$  8 (Visual Studio 2005)
  - ICC (Intel)
  - XLC (IBM)
  - Sun Studio (Sun)



# OpenMP Programming Model

- Fork-Join Parallelism

- Master thread spawns a team of threads as needed
- Parallelism added incrementally until performance goals are met: i.e. the sequential program evolves into a parallel program





# OpenMP Core Syntax

- Most of the constructs in OpenMP are compiler directives  
**#pragma omp construct [clause [clause]...]**
- Most OpenMP constructs apply to a “structured block”
  - One or more statements with one point of entry at the top and one point of exit at the bottom

```
OEGraphMol mol;  
OEParseSmiles(mol, "c1ccccc1");
```

**#pragma omp parallel**

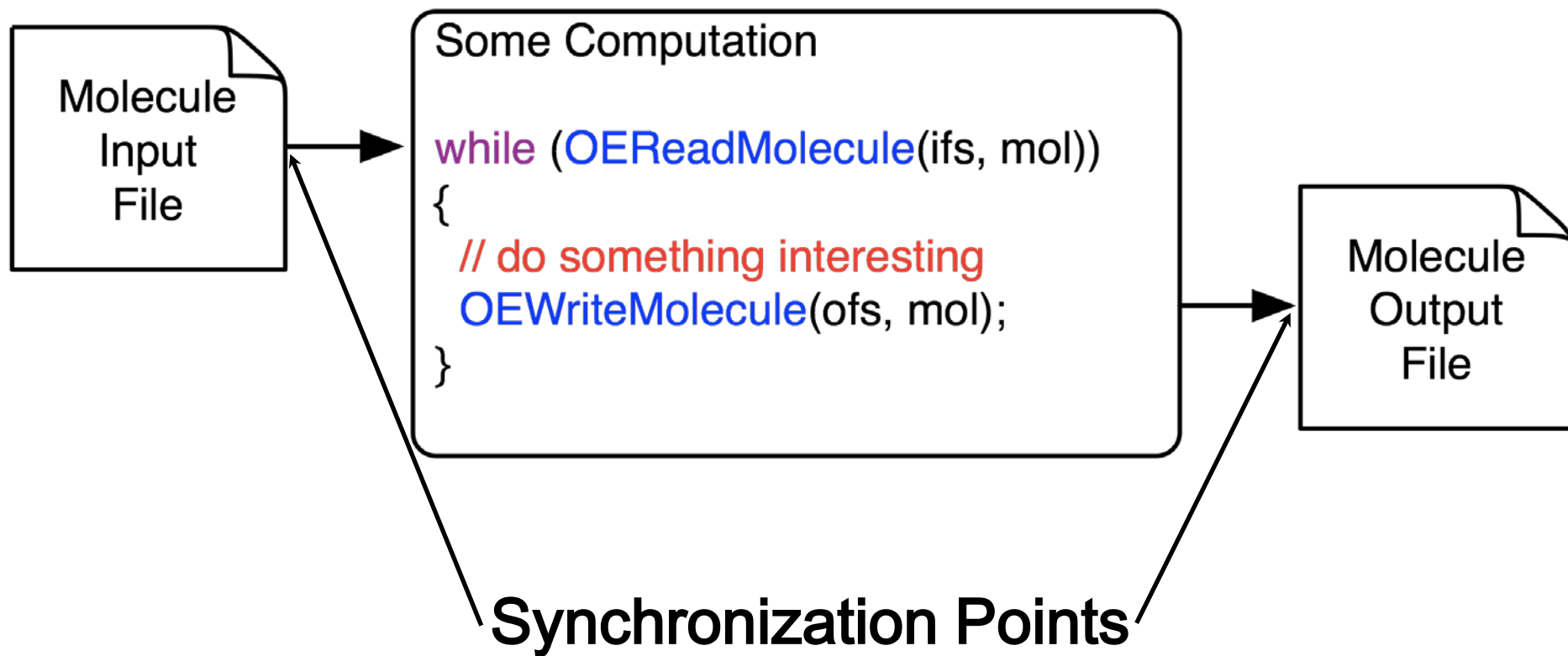
```
{  
    std::string str;  
    OECreateIsoSmiString(str, mol);
```

**#pragma omp critical**

```
    std::cout << str << std::endl;  
}
```

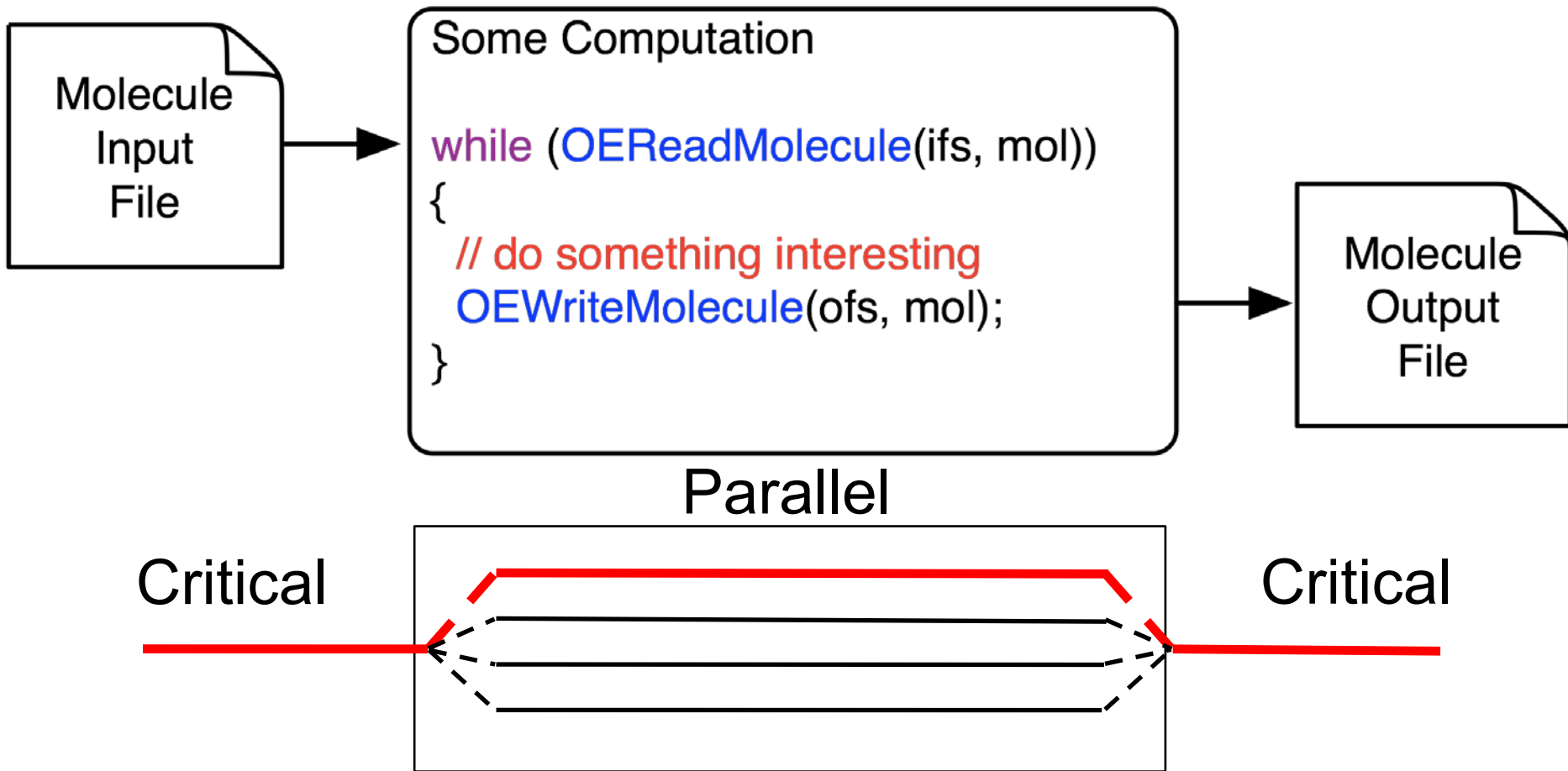


# Anatomy of an OEChem program





# Anatomy of an OEChem program





# OEChem is thread safe

“Unless otherwise specified, [OEChem] classes may safely be instantiated from multiple threads and [OEChem] functions are reentrant, but non-const use of objects of [OEChem] types is not safe if shared between threads. Use of non-constness to determine thread-safety requirements ensures consistent thread-safety specifications without having to add additional wording to each and every [OEChem] type.”



- Safe to do in multiple threads
  - Create and use OEChem types
    - OEGraphMol
  - Call OEChem functions
    - OEAddExplicitHydrogens
  - Call a “const” method of a shared object
    - OESubSearch::SingleMatch
  - Call a function with a “const” argument on a shared object
    - OECalculateMolecularWeight



- OEChem does not synchronize for you
- **NOT** safe to do in multiple threads
  - Use a non-“const” method of a shared object
    - OESubSearch.AddConstraint
  - Call a function with a non-“const” argument on a shared object
    - OEReadMolecule
    - OEWriteMolecule



```
OESubSearch pat(argv[1]);  
oemolistream ifs(argv[2]);  
oemolostream ofs(argv[3]);  
  
OEGraphMol mol;  
while(OEReadMolecule(ifs, mol))  
{  
    if(pat.SingleMatch(mol))  
    {  
        OEWriteMolecule(ofs, mol);  
    }  
}
```

Loop over a database extracting molecules that match a particular SMARTS pattern

Example Usage: molgrep c1ccccc1 drugs.sdf benzene.sdf

```

OESubSearch pat(argv[1]);
oemolistream ifs(argv[2]);
oemolostream ofs(argv[3]);

OEGraphMol mol;

while(OEReadMolecule(ifs, mol)
{
    if (pat.SingleMatch(mol))
    {
        OEWriteMolecule(ofs, mol);
    }
}

OESubSearch pat(argv[1]);
oemolistream ifs(argv[2]);
oemolostream ofs(argv[3]);

{
    OEGraphMol mol;
    OEReadMolecule(ifs, mol);
    while (mol)
    {
        if (pat.SingleMatch(mol))
        {
            OEWriteMolecule(ofs, mol);
        }
        OEReadMolecule(ifs, mol);
    }
}

```



```
OESubSearch pat(argv[1]);  
oemolistream ifs(argv[2]);  
oemolostream ofs(argv[3]);
```

```
#pragma omp parallel
```

```
{
```

```
    OEGraphMol mol;
```

```
#pragma omp critical
```

```
    OEReadMolecule(ifs, mol);
```

```
    while (mol)
```

```
    {
```

```
        if (pat.SingleMatch(mol))
```

```
        {
```

```
#pragma omp critical
```

```
            OEWwriteMolecule(ofs, mol);
```

```
        }
```

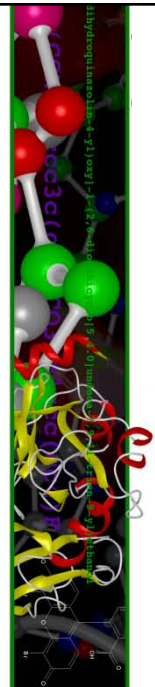
```
#pragma omp critical
```

```
    OEReadMolecule(ifs, mol);
```

```
    }
```

```
}
```

# 2x SLOWER!





# Don't forget to profile!

1. "premature optimization is the root of all evil" – Donald Knuth
2. Parallelization is used to optimize a program

Q.E.D: Premature parallelization is the root of all evil

Self	Total	Library	Symbol
0.0%	91.8%	molgrep	▼ start
0.0%	91.8%	molgrep	▼ _start
0.1%	91.8%	molgrep	▼ main
0.0%	59.8%	molgrep	▶ OEChem::OEReadMolecule(OEChem::oemolistream&, OEChem::OEGraphMol&)
0.3%	27.2%	molgrep	▶ OEChem::OEWriteMolecule(OEChem::oemolostream&, OEChem::OEMolBase&)
0.0%	4.4%	molgrep	▶ OEChem::OESubSearch::SingleMatch(OEChem::OEMolBase const&) const

## And make sure it's the sequential version!



# Amdahl's Law

$P = \% \text{ parallel}$

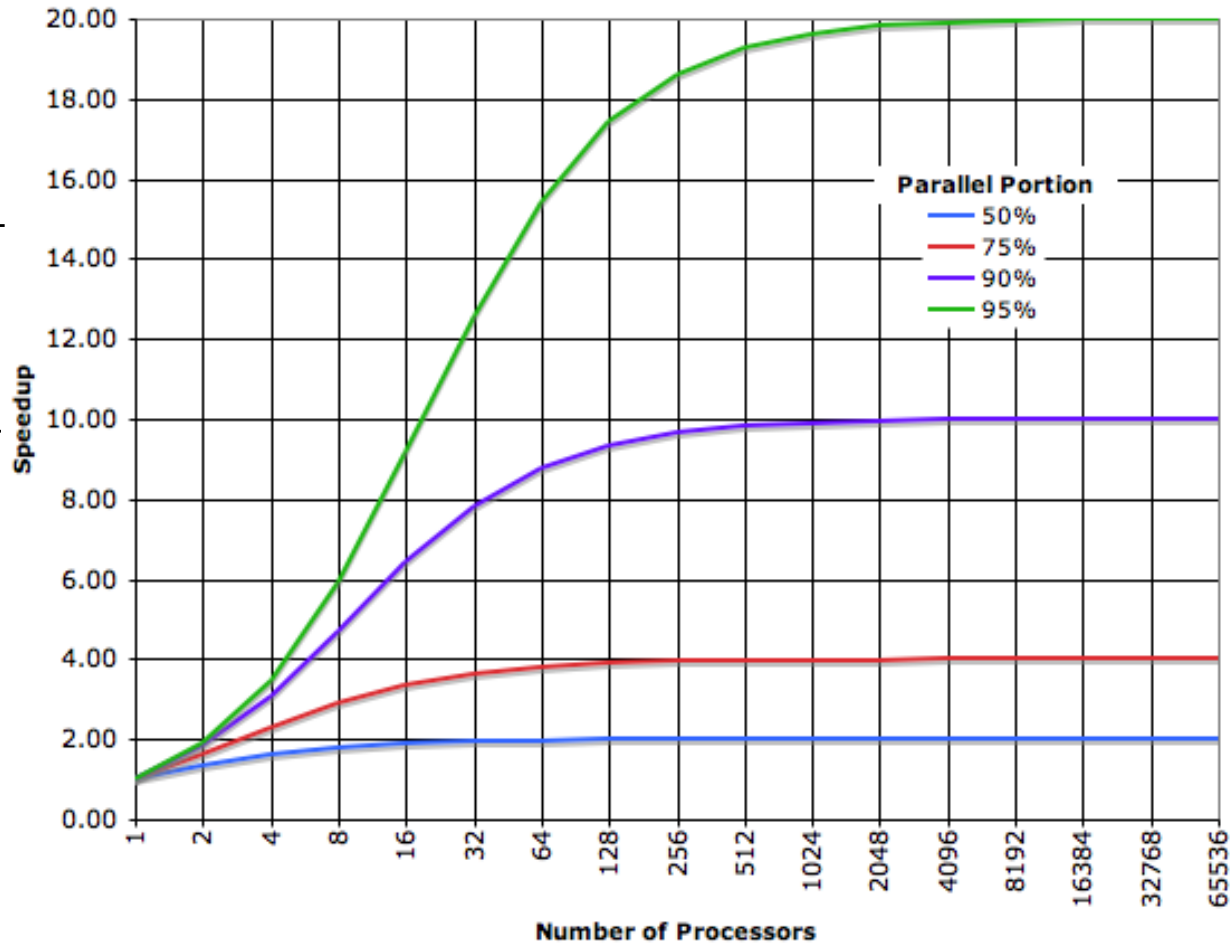
$N = \# \text{ processors}$

$$\text{Speedup} = \frac{\text{Time}_{\text{sequential}}}{\text{Time}_{\text{parallel}}}$$

$$\text{Speedup} = \frac{1}{(1 - P) + \frac{P}{N}}$$

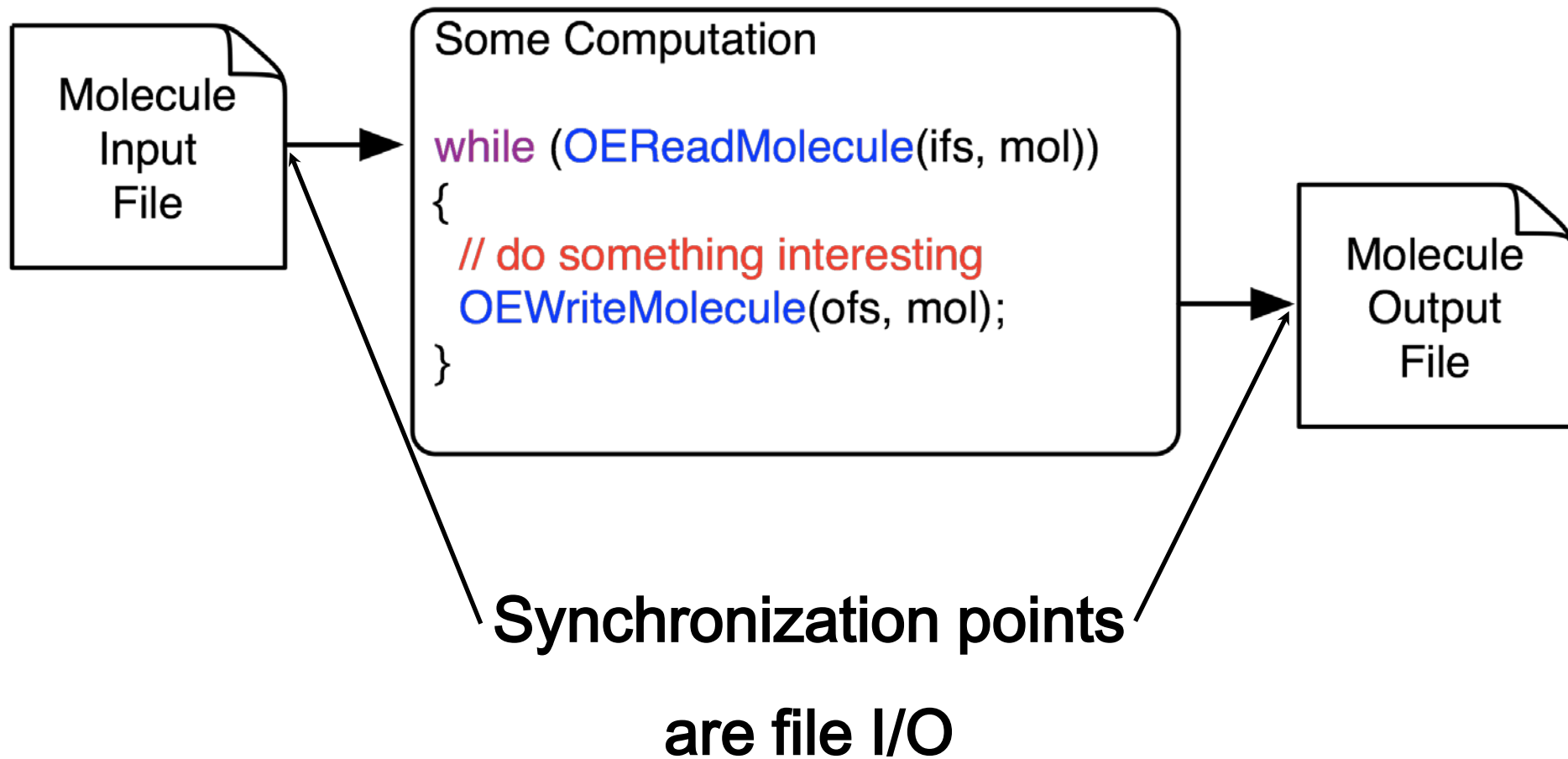
Previous attempt  
at parallelization

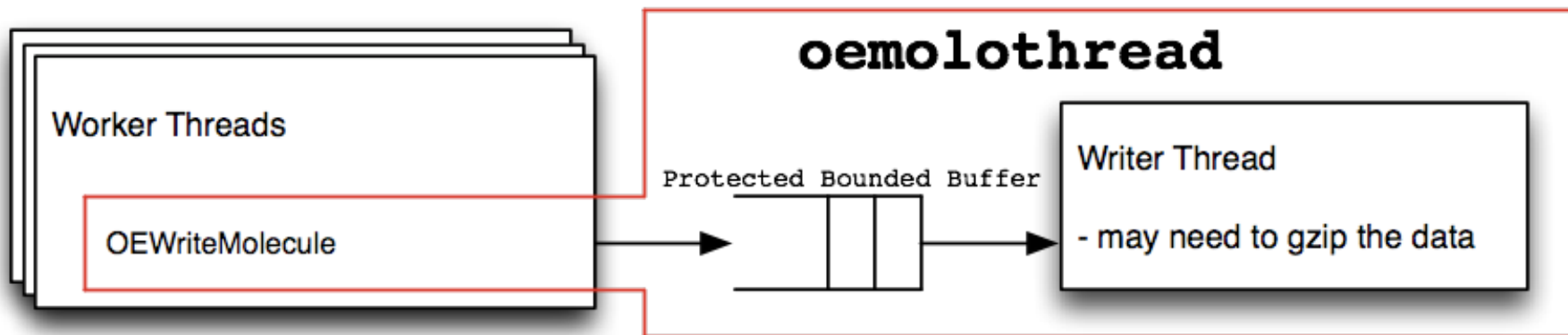
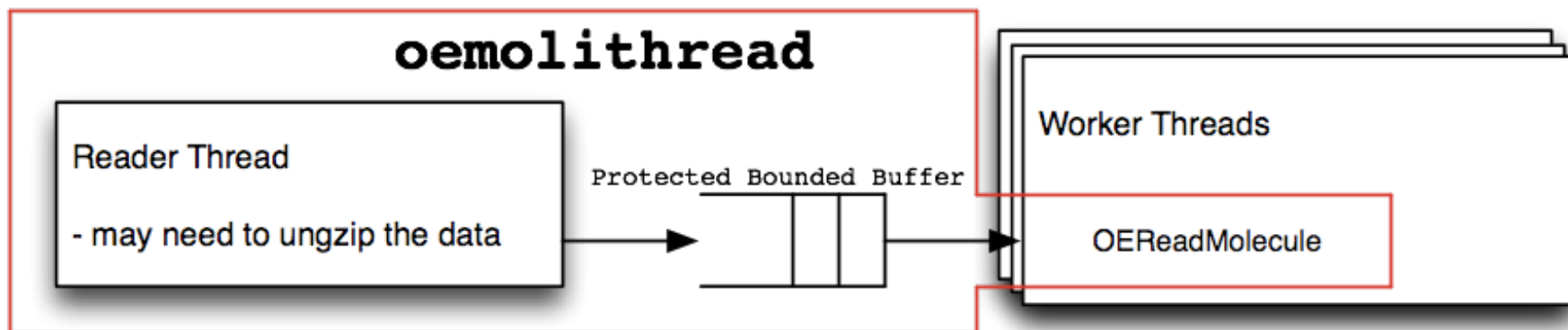
$P = \sim\%5!$





# Anatomy of an OEChem program







```
OESubSearch pat(argv[1]);  
oemolithread ifs(argv[2]);  
oemolothread ofs(argv[3]);
```

```
OEGraphMol mol;  
#pragma omp parallel  
while (OEReadMolecule(ifs, mol))  
    if (pat.SingleMatch(mol))  
        OEWwriteMolecule(ofs, mol);
```

## Crashes in Random Locations!

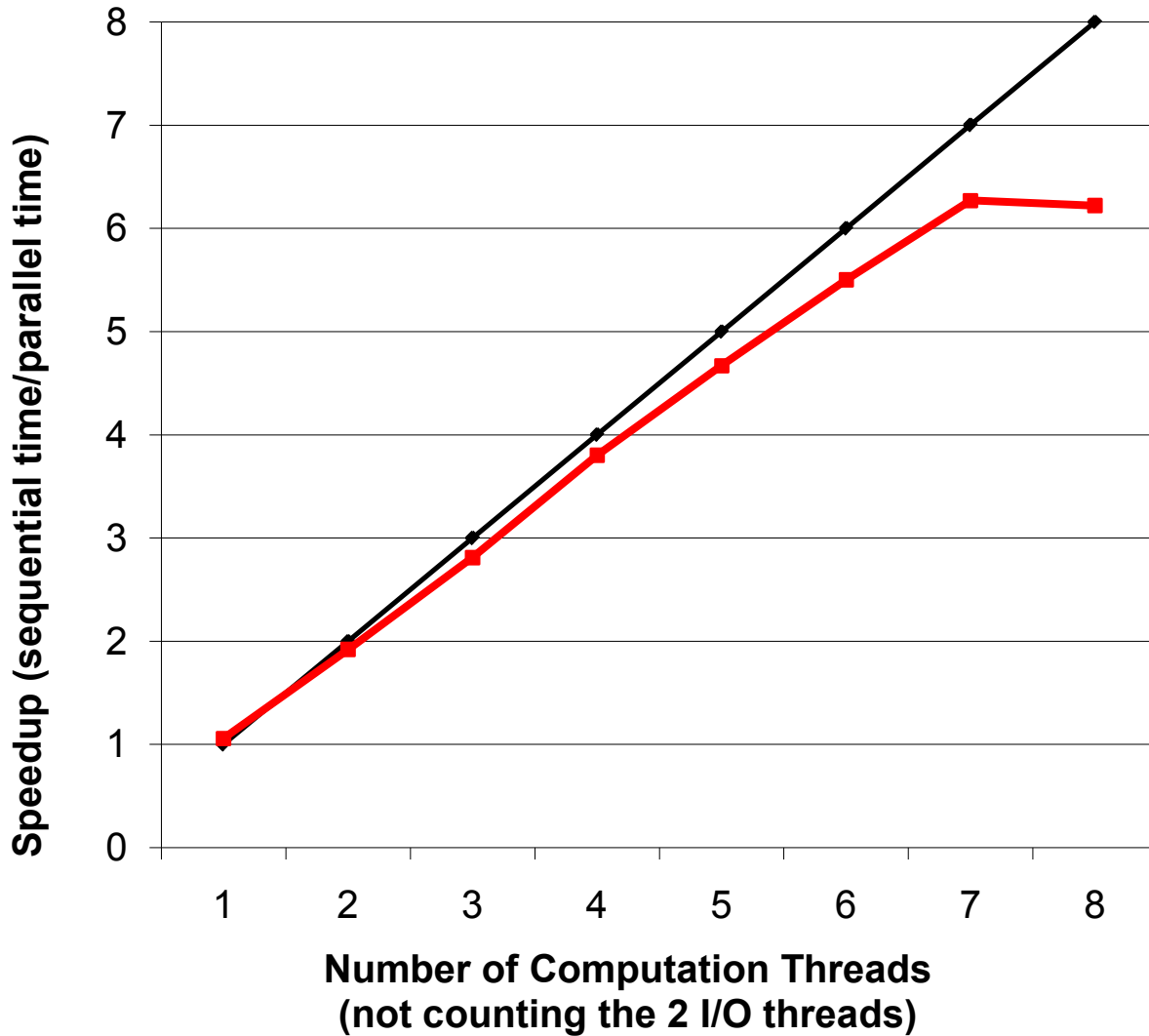


# Molecule for each thread

```
OESubSearch pat(argv[1]);  
oemolithread ifs(argv[2]);  
oemolothread ofs(argv[3]);  
  
OEGraphMol mol;  
#pragma omp parallel private(mol)  
while (OEReadMolecule(ifs, mol))  
    if (pat.SingleMatch(mol))  
        OEWriteMolecule(ofs, mol);
```



# Threaded molgrep speedup



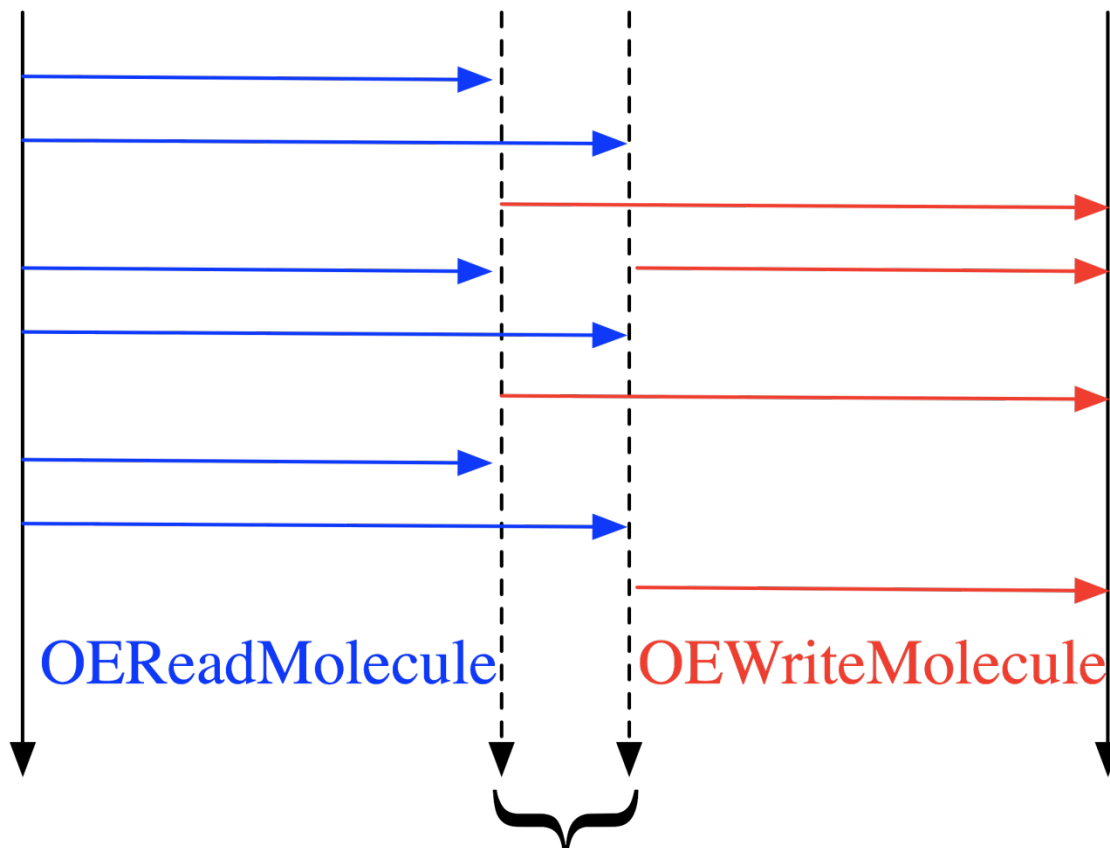
**Ideal Linear Scaling**

**Actual Speedup**



# Number of worker threads

oemolthread      Worker Threads      oemolthread



OMP\_NUM\_THREADS=2



- OpenMP is specific to C++
- Threading in Python and Java are different beasts
  - However, they can still use oemolthreads for threaded I/O

```
ss = OESubSearch(sys.argv[1])
```

```
ifs = oemolithread(sys.argv[2])
```

```
ofs = oemolothread(sys.argv[3])
```

```
class MultiCoreMolFind(Thread):
```

```
    def run(self):
```

```
        for mol in ifs.GetOEGraphMols():
```

```
            if ss.SingleMatch(mol):
```

```
                OEWriteMolecule(ofs, mol)
```

**Beware of Global Interpreter Lock!**

```
    thrs = []
```

```
    for i in xrange(OEGetNumProcessors()):
```

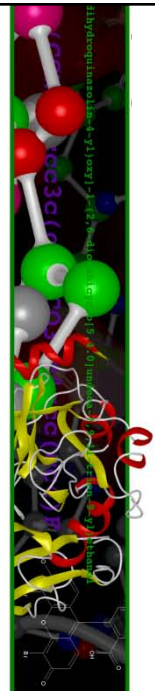
```
        thrd = MultiCoreMolFind()
```

```
        thrd.start()
```

```
        thrs.append(thrd)
```

```
    for thrd in thrs:
```

```
        thrd.join()
```



```
public static class MolFindThread extends Thread {
    public void run() {
        OEGraphMol mol = new OEGraphMol();
        while (oechem.OEReadMolecule(ifs, mol))
            if (ss.SingleMatch(mol))
                oechem.OEWriteMolecule(ofs, mol);
    }
}

public static void main(String argv[]) {
    for (int i=0; i < ncpus; i++) {
        MolFindThread thrd = new MolFindThread();
        thrd.start();
        thrds.add(thrd);
    }
    for (MolFindThread thrd : thrds) {
        try {
            thrd.join();
        } catch (InterruptedException e) {
            System.err.println(e);
        }
    }
}
```

**Beware of Garbage Collector!**





Don't clap yet!

- The best is still to come
- A demo of in-memory substructure search!



# Acknowledgements

- OpenEyers
  - Bob Tolbert
  - Geoff Skillman
  - The rest
- Intel
  - Tim Mattson
  - Ronald Green
- Temple University
  - Professor Ingargiola
  - Bencella Wisner
  - Michael Hunsberger



# MultiThreaded Substructure Search

```
unsigned int SearchMols(const OESubSearch &ss,  
                        const std::vector<OEGraphMol> &mvec)  
{  
    unsigned int count=0;  
  
    #pragma omp parallel for reduction( + : count )  
    for (int i = 0; i < (int)mvec.size(); ++i)  
    {  
        if (ss.SingleMatch(mvec[i]))  
            ++count;  
    }  
    return count;  
}
```