- import cPickle as pickle in order to transparently use the same interface.
- import job_stats from the tacc_stats monitor directory

In [1]:
```
import sys
sys.path.append('../../monitor')
import job_stats
import cPickle as pickle
```

- Load a single job's file from my directory of them on Ranger
- This is a local untar of John's nightly files

In [2]:
```
j=pickle.load(open('nightly_jobs/2012-10-30/2887373'))
```

- Let's see what's actually in this object.

In [3]:
```
dir(j)
```

Out[3]:
```
['__class__',
 '__delattr__',
 '__doc__',
 '__format__',
 '__getattribute__',
 '__hash__',
 '__init__',
 '__module__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__slots__',
 '__str__',
 '__subclasshook__',
 'acct',
 'aggregate_stats',
 'end_time',
 'error',
 'gather_stats',
 'get_schema',
 'get_stats',
 'hosts',
 'id',
 'munge_times',
 'process_dev_stats',
 'process_stats',
 'schemas',
 'start_time',
 'times',
 'trace']
```

- The hosts data structure contains most of the actual data
- But we'll come back to a few others

In [4]:
```
j.hosts
```

Out[4]:
```
{'i101-111.ranger.tacc.utexas.edu': <job_stats.Host at 0x10204eed0>,
 'i112-306.ranger.tacc.utexas.edu': <job_stats.Host at 0x10b2e10d0>,
 'i113-106.ranger.tacc.utexas.edu': <job_stats.Host at 0x10b2e1110>,
 'i113-108.ranger.tacc.utexas.edu': <job_stats.Host at 0x10204ee10>,
 'i119-101.ranger.tacc.utexas.edu': <job_stats.Host at 0x10b2e1090>,
 'i133-403.ranger.tacc.utexas.edu': <job_stats.Host at 0x101dbe3d0>,
 'i137-102.ranger.tacc.utexas.edu': <job_stats.Host at 0x101dbb8d0>,
 'i145-106.ranger.tacc.utexas.edu': <job_stats.Host at 0x101e5bb10>,
 'i149-102.ranger.tacc.utexas.edu': <job_stats.Host at 0x101dbe410>,
 'i149-311.ranger.tacc.utexas.edu': <job_stats.Host at 0x10204ef50>,
 'i152-401.ranger.tacc.utexas.edu': <job_stats.Host at 0x10204edd0>,
 'i154-410.ranger.tacc.utexas.edu': <job_stats.Host at 0x101dbe510>,
 'i158-310.ranger.tacc.utexas.edu': <job_stats.Host at 0x10204ee50>,
 'i162-207.ranger.tacc.utexas.edu': <job_stats.Host at 0x10204ef10>,
 'i168-212.ranger.tacc.utexas.edu': <job_stats.Host at 0x10204efd0>,
 'i169-211.ranger.tacc.utexas.edu': <job_stats.Host at 0x101dbbe10>,
 'i176-109.ranger.tacc.utexas.edu': <job_stats.Host at 0x10b2e1050>,
 'i176-112.ranger.tacc.utexas.edu': <job_stats.Host at 0x10204ee90>,
 'i181-306.ranger.tacc.utexas.edu': <job_stats.Host at 0x10204ef90>}
```

- Hosts is a dictionary of further objects.
- Let's pick one and have a look around

In [5]:
```
h='i101-111.ranger.tacc.utexas.edu'
dir(j.hosts[h])
```

Out[5]:
```
['__class__',
 '__delattr__',
 '__dict__',
 '__doc__',
 '__format__',
 '__getattribute__',
 '__hash__',
 '__init__',
 '__module__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
 'error',
 'gather_stats',
 'get_stats',
 'get_stats_paths',
 'job',
 'marks',
 'name',
 'parse_stats',
 'read_stats_file',
 'read_stats_file_header',
 'stats',
 'trace']
```

- The stats dictionary contains all of the data for this host

In [6]:
```
j.hosts[h].stats.keys()
```

Out[6]:
```
['ps',
 'amd64_sock',
 'lnet',
 'mem',
 'llite',
 'vm',
 'sysv_shm',
 'amd64_core',
 'numa',
 'net',
 'tmpfs',
 'vfs',
 'cpu',
 'block',
 'ib_sw']
```

- Each key here corresponds (more or less) to the lines from the raw data
- The value for each key is a dictionary containing key/value pairs that point to numpy arrays of the actual data for this type.
- In the case of amd64_core, there's a dictionary entry for each core (strings '0' through '15' on Ranger)

In [7]:
```
j.hosts[h].stats['amd64_core']['0']
```

Out[7]:
```
array([[             0,              0,              0],
       [   672322486793,    244443211072,     23836512957],
       [  1582905469054,    570433271488,     57033734781],
       [  2630260890974,    772689561088,     72806955413],
       [  3544111028919,   1056275726016,     94484478062],
       [  4469708202217,   1339141086272,    116993344068],
       [  5403724140262,   1622699744064,    140416884014],
       [  6339758624511,   1907506659072,    164308891339],
       [  7278085528861,   2164789184896,    184305633969],
       [  8223626657934,   2433906842368,    205218923263],
       [  9155480787591,   2710674718336,    226568975083],
       [ 10099406145982,   2991129969536,    249877419073],
       [ 11031855201532,   3253103970304,    270527680112],
       [ 11978492245378,   3522650667392,    291467237077],
       [ 12931241269100,   3792735238144,    313293140007],
       [ 13877399003702,   4071925403776,    336508041477],
       [ 14817008040545,   4333454289984,    357992006775],
       [ 15755694983207,   4607378583616,    379178111515],
       [ 16689877183254,   4886062543232,    401003272674],
       [ 17641752260134,   5166123192192,    424697788352],
       [ 18599052783044,   5439558904256,    447578226683],
       [ 19531135916388,   5697525862272,    467208752172],
       [ 20462672918290,   5974731499136,    488585464240],
       [ 21410756621530,   6253488031488,    511800133446],
       [ 22355623050282,   6510835389632,    533044554937],
       [ 23297734837795,   6783576448128,    554174603954],
       [ 24245171225946,   7058298770880,    576288959216],
       [ 25180973207471,   7342083936256,    599796891201],
       [ 26113161144089,   7606714535040,    620612696985],
       [ 27048810221292,   7882293906176,    641902657767],
       [ 27999162256959,   8160595163840,    665094588944],
       [ 28922810328250,   8430981315584,    686540868604],
       [ 29859976319564,   8708150395264,    708395167192],
       [ 30804794073287,   8983069847808,    730599101169],
       [ 31745121433107,   9254894721536,    752778153174],
       [ 32672271371667,   9530151017664,    774810459287],
       [ 33605178473514,   9807339687488,    796211334444],
       [ 34560034791350,  10081698405568,    819180713197],
       [ 35487618120062,  10347569215424,    840090212933],
       [ 36417962286177,  10624208840320,    861444700425],
       [ 37351781939016,  10898945002688,    882681660275],
       [ 38301911411267,  11174700467136,    905701392186],
       [ 39245779520018,  11453439199232,    928885469720],
       [ 40196607931819,  11724025806656,    951572953447],
       [ 41154326981092,  11971335575552,    970512446915],
       [ 42106465603804,  12239052142848,    991327331845],
       [ 43060337917066,  12505219714880,   1012046350098],
       [ 44042799691375,  12765899584000,   1034164628272],
       [ 45001062940950,  13011196735808,   1052981571296],
       [ 45950826837754,  13279380818496,   1073830116063],
       [ 46905094973278,  13545048903104,   1094524279796],
       [ 47878639010788,  13804359927616,   1115702682066],
       [ 48846853321243,  14047296410048,   1135233704355],
       [ 49802458502478,  14313996702528,   1156012315681],
       [ 50773630761852,  14577302469440,   1177463371208],
       [ 51750458805180,  14835774389312,   1198617815583],
       [ 52739478013711,  15080810576128,   1219150608258],
       [ 53697957020474,  15345714552960,   1239822813201],
       [ 54654841695832,  15610873138816,   1260417321873],
       [ 55626978268439,  15871184033472,   1281344717506],
       [ 56608356014892,  16128617872000,   1302888854736],
       [ 57596587582005,  16385920936832,   1324757509346],
       [ 58972704627927,  16386082136704,   1324757509346],
       [ 60347539810597,  16386234461888,   1324757509346],
```

- These three columns represent the three core counters we count: SSE_FLOPS, Data Cache Sys Fills, and User Cycles
- There is one row for each time sample
- How do we know which column corresponds to which counter?

- The get_schema method returns the schema for this data for a given initial key

In [8]:
```
j.get_schema('amd64_core')
```

Out[8]:
```
{'USER': (is_event=True), 'DCSF': (is_event=True, unit=B), 'SSE_FLOPS': (is_event=True)}
```

- The keys of this dictionary let you know what's available
- There are some additional things like the unit of measure for a thing, and whether it's an event or not

- To get the column index into the host stats array, you can do:

In [9]:
```
index=j.get_schema('amd64_core')['SSE_FLOPS'].index
print index
```

```
2
```

- So now we can do:

In [10]:
```
v=j.hosts[h].stats['amd64_core']['0'][:,index]
print v
```

```
[            0   23836512957   57033734781   72806955413   94484478062
  116993344068  140416884014  164308891339  184305633969  205218923263
  226568975083  249877419073  270527680112  291467237077  313293140007
  336508041477  357992006775  379178111515  401003272674  424697788352
  447578226683  467208752172  488585464240  511800133446  533044554937
  554174603954  576288959216  599796891201  620612696985  641902657767
  665094588944  686540868604  708395167192  730599101169  752778153174
  774810459287  796211334444  819180713197  840090212933  861444700425
  882681660275  905701392186  928885469720  951572953447  970512446915
  991327331845 1012046350098 1034164628272 1052981571296 1073830116063
 1094524279796 1115702682066 1135233704355 1156012315681 1177463371208
 1198617815583 1219150608258 1239822813201 1260417321873 1281344717506
 1302888854736 1324757509346 1324757509346 1324757509346 1324757509346
 1324757509346 1324757509346 1324757509346 1324757509346 1324757509346
 1324757509346 1324757509346 1324757509346 1324757509346 1324757509346
 1324757509346 1324757509346 1324757509346 1324757509346 1324757509346
 1324757509346 1324757509349 1324757509349 1324757509349 1324757509349
 1324757509349 1324757509349 1324757509349 1324757509349 1324757509349
 1324757509349 1324757509349 1324757509349 1324757509349 1324757509349
 1324757509349 1324757509349 1324757509350 1324757509350 1324757509350
 1324757509350 1324757509350 1324757509350 1324757509350 1324757509350
 1324757509350 1324757509350 1324757509350 1324757509350 1324757509350
 1324757509350 1324757509350 1324757509350 1324757509350 1324757509350
 1324757509350 1324757509350 1324757509350 1324757509350 1324757509350
 1324757509350 1324757509350 1324757509350 1324757509350 1324757509350
 1324757509350 1324757509350 1324757509350 1324757509350 1324757509350
 1324757509350 1324757509350 1324757509350 1324757509350 1324757509352
 1324757509352 1324757509352 1324757509352 1324757509352 1324757509352
 1324757509356 1324757509356 1324757509356 1324757509356 1324757509356
 1324758011074]
```

- Which is a 1-D numpy array of values for SSE_FLOPS for core 0 on host i101-111 on Ranger

- These values are time-aligned and robustified in various ways (account for roll-over, etc.).
- They count raw SSE floating point operations as measured by the hardware counters.
- They are more useful than the raw data, but typically what we want is rates:

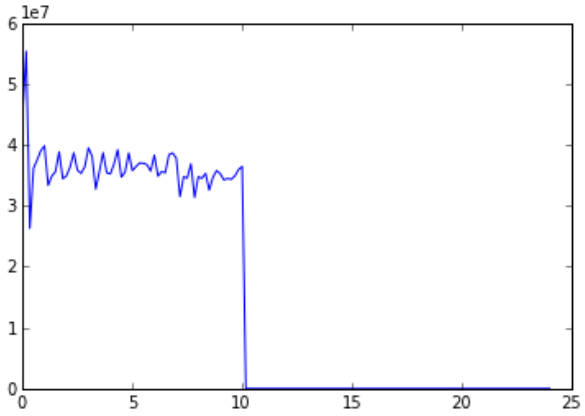
In [11]:
```
import numpy
r = numpy.diff(v)/numpy.diff(j.times)
print r
```

```
[46194792 55328703 26288701 36129204 37514776 39039233 39820012 33327904
 34855482 35583419 38847406 34417101 34899261 36376504 38691502 35806608
 35310174 36375268 39490859 38134063 32717542 35627853 38691115 35407369
 35216748 36857258 39179886 34693009 35483267 38653218 35743799 36423830
 37006556 36965086 36720510 35668125 38282297 34849166 35590812 35394933
 38366219 38640129 37812472 31513300 34749390 34531697 36863796 31361571
 34747574 34490272 35297337 32551703 34631018 35751759 35257407 34221321
 34453674 34324181 34878992 35906895 36447757        0        0        0
        0        0        0        0        0        0        0        0
        0        0        0        0        0        0        0        0
        0        0        0        0        0        0        0        0
        0        0        0        0        0        0        0        0
        0        0        0        0        0        0        0        0
        0        0        0        0        0        0        0        0
        0        0        0        0        0        0        0        0
        0        0        0        0        0        0        0        0
        0        0        0        0        0        0        0        0
     3413]
```

In [12]:
```
%pylab inline
from pylab import *
t=j.times-j.times[0]
ax=subplot(111)
plot(t[0:-1]/3600.,r)
```

```
Welcome to pylab, a matplotlib-based Python environment [backend: module://IPython.zmq.pylab.bac
For more information, type 'help(pylab)'.
```

Out[12]:
```
[<matplotlib.lines.Line2D at 0x10b2e1c10>]
```



- There is a simpler interface to the data
- But you still need to look at the schema to know what's there

In [13]:
```
help(j.hosts[h].get_stats)
```

```
Help on method get_stats in module job_stats:

get_stats(self, type_name, dev_name, key_name) method of job_stats.Host instance
    Host.get_stats(type_name, dev_name, key_name)
    Return the vector of stats for the given type, dev, and key.
```

In [14]:
```
j.hosts[h].get_stats('amd64_core','0','SSE_FLOPS')
```

Out[14]:
```
array([            0,   23836512957,   57033734781,   72806955413,
        94484478062,  116993344068,  140416884014,  164308891339,
       184305633969,  205218923263,  226568975083,  249877419073,
       270527680112,  291467237077,  313293140007,  336508041477,
       357992006775,  379178111515,  401003272674,  424697788352,
       447578226683,  467208752172,  488585464240,  511800133446,
       533044554937,  554174603954,  576288959216,  599796891201,
       620612696985,  641902657767,  665094588944,  686540868604,
       708395167192,  730599101169,  752778153174,  774810459287,
       796211334444,  819180713197,  840090212933,  861444700425,
       882681660275,  905701392186,  928885469720,  951572953447,
       970512446915,  991327331845, 1012046350098, 1034164628272,
      1052981571296, 1073830116063, 1094524279796, 1115702682066,
      1135233704355, 1156012315681, 1177463371208, 1198617815583,
      1219150608258, 1239822813201, 1260417321873, 1281344717506,
      1302888854736, 1324757509346, 1324757509346, 1324757509346,
      1324757509346, 1324757509346, 1324757509346, 1324757509346,
      1324757509346, 1324757509346, 1324757509346, 1324757509346,
      1324757509346, 1324757509346, 1324757509346, 1324757509346,
      1324757509346, 1324757509346, 1324757509346, 1324757509346,
      1324757509346, 1324757509349, 1324757509349, 1324757509349,
      1324757509349, 1324757509349, 1324757509349, 1324757509349,
      1324757509349, 1324757509349, 1324757509349, 1324757509349,
      1324757509349, 1324757509349, 1324757509349, 1324757509349,
      1324757509349, 1324757509350, 1324757509350, 1324757509350,
      1324757509350, 1324757509350, 1324757509350, 1324757509350,
      1324757509350, 1324757509350, 1324757509350, 1324757509350,
      1324757509350, 1324757509350, 1324757509350, 1324757509350,
      1324757509350, 1324757509350, 1324757509350, 1324757509350,
      1324757509350, 1324757509350, 1324757509350, 1324757509350,
      1324757509350, 1324757509350, 1324757509350, 1324757509350,
      1324757509350, 1324757509350, 1324757509350, 1324757509350,
      1324757509350, 1324757509350, 1324757509352, 1324757509352,
      1324757509352, 1324757509352, 1324757509352, 1324757509352,
      1324757509356, 1324757509356, 1324757509356, 1324757509356,
      1324757509356, 1324758011074], dtype=uint64)
```

j.get_stats('amd64_core','0','SSE_FLOPS')

In [18]:
```
j.acct['account']
```

Out[18]:
```
'TG-DMR090026'
```

In [19]:
```
j.acct.keys()
```

Out[19]:
```
['ru_isrss',
 'ar_submission_time',
 'ru_inblock',
 'io',
 'owner',
 'slots',
 'id',
 'category',
 'queue',
 'ru_oublock',
 'group',
 'exit_status',
 'ru_stime',
 'ru_nsignals',
 'hostname',
 'arid',
 'priority',
 'failed',
 'ru_ixrss',
 'ru_nivcsw',
 'department',
 'pe_taskid',
 'task_number',
 'granted_pe',
 'mem',
 'start_time',
 'ru_msgsnd',
 'ru_wallclock',
 'ru_minflt',
 'submission_time',
 'maxvmem',
 'ru_nvcsw',
 'ru_nswap',
 'ru_majflt',
 'account',
 'iow',
 'name',
 'project',
 'ru_utime',
 'ru_ismrss',
 'end_time',
 'ru_idrss',
 'ru_maxrss',
 'ru_msgrcv',
 'cpu']
```

In [20]:
```
j.acct['submission_time']
```

Out[20]:
```
1351525419
```

In [ ]:
```

```