Weak versus strong scaling

• **Strong scaling:** *Fixed total* problem size as we run on more processes
  - Sorting $n$ numbers on 1 process, 2 processes, 4 processes, ...

• **Weak scaling:** Fixed problem size per process but *increasing total* problem size as we run on more processes
  - Sorting $n$ numbers on 1 process
  - $2n$ numbers on 2 processes
  - $4n$ numbers on 4 processes
Amdahl’s law

• Speedup is limited by the serial portion of the code
  • Often referred to as the serial “bottleneck”

• Lets say only a fraction $f$ of the code can be parallelized on $p$ processes

$$\text{Speedup} = \frac{1}{(1 - f) + f/p}$$
Amdahl’s law

- Speedup is limited by the serial portion of the code
  - Often referred to as the serial “bottleneck”
- Lets say only a fraction $f$ of the code can be parallelized on $p$ processes

\[
\text{Speedup} = \frac{1}{(1 - f) + \frac{f}{p}}
\]
Amdahl’s law

- Speedup is limited by the serial portion of the code
  - Often referred to as the serial “bottleneck”
- Lets say only a fraction $f$ of the code can be parallelized on $p$ processes

\[
\text{Speedup} = \frac{1}{(1 - f) + \frac{f}{p}}
\]
Performance analysis

- Parallel performance of a program might not be what the developer expects
- How do we find performance bottlenecks?
- Performance analysis is the process of studying the performance of parallel code
- Identify why performance might be slow
  - Serial performance
  - Serial bottlenecks when running in parallel
  - Communication overheads
Performance analysis methods

- Analytical techniques: use algebraic formulae
  - In terms of data size (n), number of processes (p)
- Time complexity analysis
- Scalability analysis (Isoefficiency)
- Model performance of various operations
  - Analytical models: LogP, alpha-beta model
- Empirical performance analysis using tools
Parallel prefix sum
Parallel prefix sum
Parallel prefix sum for $n >> p$

- Assign $n/p$ elements (block) to each process
- Perform prefix sum on these blocks on each process locally
  - Number of calculations:
- Then do parallel algorithm with partial prefix sums
  - Number of phases:
- Total number of calculations:
Parallel prefix sum for $n \gg p$

- Assign $n/p$ elements (block) to each process
- Perform prefix sum on these blocks on each process locally
  - Number of calculations: $\frac{n}{p}$
- Then do parallel algorithm with partial prefix sums
  - Number of phases:
- Total number of calculations:
Parallel prefix sum for $n >> p$

- Assign $n/p$ elements (block) to each process
- Perform prefix sum on these blocks on each process locally
  - Number of calculations: $\frac{n}{p}$
- Then do parallel algorithm with partial prefix sums
  - Number of phases: $\log(p)$
- Total number of calculations:
Parallel prefix sum for $n \gg p$

- Assign $n/p$ elements (block) to each process

- Perform prefix sum on these blocks on each process locally
  - Number of calculations: $\frac{n}{p}$

- Then do parallel algorithm with partial prefix sums
  - Number of phases: $\log(p)$

- Total number of calculations: $\log(p) \times \frac{n}{p}$
Modeling communication: LogP model

- Model for communication on an interconnection network

L: latency or delay

o: overhead (processor busy in communication)

g: gap

P: number of processors / processes

\[ \frac{1}{g} = \text{bandwidth} \]
alpha + n * beta model

- Another model for communication

\[ T_{\text{comm}} = \alpha + n \times \beta \]

\( \alpha \): latency

\( n \): size of message

\( 1/\beta \): bandwidth
Isoefficiency

- Relationship between problem size and number of processors to maintain a certain level of efficiency

- At what rate should we increase problem size with respect to number of processors to keep efficiency constant
Speedup and efficiency

- **Speedup**: Ratio of execution time on one process to that on $p$ processes

  \[
  \text{Speedup} = \frac{t_1}{t_p}
  \]

- **Efficiency**: Speedup per process

  \[
  \text{Efficiency} = \frac{t_1}{t_p \times p}
  \]
Efficiency in terms of overhead

- Total time spent in all processes = (useful) computation + overhead (extra computation + communication + idle time)

\[ p \times t_p = t_1 + t_o \]

Efficiency = \[ \frac{t_1}{t_p \times p} = \frac{t_1}{t_1 + t_o} = \frac{1}{1 + \frac{t_o}{t_1}} \]
Isoefficiency function

Efficiency: \[ \text{Efficiency} = \frac{1}{1 + \frac{t_o}{t_1}} \]

- Efficiency is constant if \( t_o / t_1 \) is constant (\( K \))

\[ t_o = K \times t_1 \]
Isoefficiency analysis

- 1D decomposition:
  - Computation:
  - Communication:

- 2D decomposition:
  - Computation:
  - Communication
Isoefficiency analysis

- **1D decomposition:**
  - Computation: $\sqrt{n} \times \frac{\sqrt{n}}{p} = \frac{n}{p}$
  - Communication:

- **2D decomposition:**
  - Computation:
  - Communication
Isoefficiency analysis

- 1D decomposition:
  - Computation: $\sqrt{n} \times \frac{\sqrt{n}}{p} = \frac{n}{p}$
  - Communication: $2 \times \sqrt{n}$

- 2D decomposition:
  - Computation:
  - Communication
Isoefficiency analysis

• 1D decomposition:
  • Computation: \[ \sqrt{n} \times \frac{\sqrt{n}}{p} = \frac{n}{p} \]
  • Communication: \[ 2 \times \sqrt{n} \]
  \[ \frac{t_0}{t_1} = \frac{2 \times \sqrt{n}}{\frac{n}{p}} = \frac{2 \times p}{\sqrt{n}} \]

• 2D decomposition:
  • Computation:
  • Communication
Isoefficiency analysis

- **1D decomposition:**
  - Computation: \( \sqrt{n} \times \frac{\sqrt{n}}{p} = \frac{n}{p} \)
  - Communication: \( 2 \times \sqrt{n} \)

- **2D decomposition:**
  - Computation: \( \frac{\sqrt{n}}{\sqrt{p}} \times \frac{\sqrt{n}}{\sqrt{p}} = \frac{n}{p} \)
  - Communication

\[
\frac{t_0}{t_1} = \frac{2 \times \sqrt{n}}{\frac{n}{p}} = \frac{2 \times p}{\sqrt{n}}
\]
Isoefficiency analysis

- 1D decomposition:
  - Computation: \( \sqrt{n} \times \frac{\sqrt{n}}{p} = \frac{n}{p} \)
  - Communication: \( 2 \times \sqrt{n} \)

\[
\frac{t_0}{t_1} = \frac{2 \times \sqrt{n}}{\frac{n}{p}} = \frac{2 \times p}{\sqrt{n}}
\]

- 2D decomposition:
  - Computation: \( \frac{\sqrt{n}}{\sqrt{p}} \times \frac{\sqrt{n}}{\sqrt{p}} = \frac{n}{p} \)
  - Communication: \( 4 \times \frac{\sqrt{n}}{\sqrt{p}} \)
 Isoefficiency analysis

• 1D decomposition:
  • Computation: \( \sqrt{n} \times \frac{\sqrt{n}}{p} = \frac{n}{p} \)
  • Communication: \( 2 \times \sqrt{n} \)
  \[
  \frac{t_0}{t_1} = \frac{2 \times \sqrt{n}}{\frac{n}{p}} = \frac{2 \times p}{\sqrt{n}}
  \]

• 2D decomposition:
  • Computation: \( \frac{\sqrt{n}}{\sqrt{p}} \times \frac{\sqrt{n}}{\sqrt{p}} = \frac{n}{p} \)
  • Communication \( 4 \times \frac{\sqrt{n}}{\sqrt{p}} \)
  \[
  \frac{t_0}{t_1} = \frac{4 \times \frac{\sqrt{n}}{\sqrt{p}}}{\frac{n}{p}} = \frac{4 \times \sqrt{p}}{\sqrt{n}}
  \]
Empirical performance analysis

- Two parts to performance analysis
  - measurement
  - analysis/visualization
- Simplest tool: timers in the code and printf
Using timers

double start, end;
double phase1, phase2, phase3;

start = MPI_Wtime();
    ... phase1 code ...
end = MPI_Wtime();
phase1 = end - start;

start = MPI_Wtime();
    ... phase2 ...
end = MPI_Wtime();
phase2 = end - start;

start = MPI_Wtime();
    ... phase3 ...
end = MPI_Wtime();
phase3 = end - start;
double start, end;
double phase1, phase2, phase3;

start = MPI_Wtime();
    ... phase1 code ...
end = MPI_Wtime();
phase1 = end - start;

start = MPI_Wtime();
    ... phase2 ...
end = MPI_Wtime();
phase2 = end - start;

start = MPI_Wtime();
    ... phase3 ...
end = MPI_Wtime();
phase3 = end - start;

Phase 1 took 2.45 s
Phase 2 took 11.79 s
Phase 3 took 4.37 s
Performance tools

- **Tracing tools**
  - Capture entire execution trace

- **Profiling tools**
  - Provide aggregated information
  - Typically use statistical sampling

- Many tools can do both
Metrics recorded

- Counts of function invocations
- Time spent in code
- Number of bytes sent
- Hardware counters
- To fix performance problems — we need to connect metrics to source code
Tracing tools

- Record all the events in the program with timestamps
- Events: function calls, MPI events, etc.

Tracing tools

• Record all the events in the program with timestamps
• Events: function calls, MPI events, etc.

Tracing tools

- Record all the events in the program with timestamps
- Events: function calls, MPI events, etc.

Examples of tracing tools

- VampirTrace
- Score-P
- TAU
- Projections
- HPCToolkit
Profiling tools

- Ignore the specific times at which events occurred
- Provide aggregate information about different parts of the code
- Examples:
  - Gprof, perf
  - mpiP
  - HPCToolkit, caliper
- Python tools: cprofile, pyinstrument, scalene

Gprof data in hpctView
Calling contexts, trees, and graphs

- Calling context or call path: Sequence of function invocations leading to the current sample
- Calling context tree (CCT): dynamic prefix tree of all call paths in an execution
- Call graph: merge nodes in a CCT with the same name into a single node but keep caller-callee relationships as arcs
Calling context trees, call graphs, …
Calling context trees, call graphs, …

Calling context tree (CCT)
Calling context trees, call graphs, ...

Contextual information

- File
- Line number
- Function name
- Callpath
- Load module
- Process ID
- Thread ID

Calling context tree (CCT)
Calling context trees, call graphs, ...

Contextual information
- File
- Line number
- Function name
- Callpath
- Load module
- Process ID
- Thread ID

Performance Metrics
- Time
- Flops
- Cache misses

Calling context tree (CCT)
Calling context trees, call graphs, ...

**Contextual information**
- File
- Line number
- Function name
- Callpath
- Load module
- Process ID
- Thread ID

**Performance Metrics**
- Time
- Flops
- Cache misses
Output of profiling tools

- Flat profile: Listing of all functions with counts and execution times
- Call graph profile
- Calling context tree
Hatchet

- Hatchet enables programmatic analysis of parallel profiles
- Leverages pandas which supports multi-dimensional tabular datasets
- Create a structured index to enable indexing pandas dataframes by nodes in a graph
- A set of operators to filter, prune and/or aggregate structured data

https://hatchet.readthedocs.io/en/latest/
Pandas and dataframes
Pandas and dataframes

- Pandas is an open-source Python library for data analysis
Pandas and dataframes

- Pandas is an open-source Python library for data analysis
- Dataframe: two-dimensional tabular data structure
- Supports many operations borrowed from SQL databases

```
<table>
<thead>
<tr>
<th>node</th>
<th>name</th>
<th>time (inc)</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>{&quot;name&quot;: 'main'}</td>
<td>main</td>
<td>200.0</td>
<td>10.0</td>
</tr>
<tr>
<td>{&quot;name&quot;: 'physics'}</td>
<td>physics</td>
<td>60.0</td>
<td>40.0</td>
</tr>
<tr>
<td>{&quot;name&quot;: 'mpi'}</td>
<td>mpi</td>
<td>20.0</td>
<td>5.0</td>
</tr>
<tr>
<td>{&quot;name&quot;: 'psm2'}</td>
<td>psm2</td>
<td>15.0</td>
<td>30.0</td>
</tr>
<tr>
<td>{&quot;name&quot;: 'solvers'}</td>
<td>solvers</td>
<td>100.0</td>
<td>10.0</td>
</tr>
<tr>
<td>{&quot;name&quot;: 'hypre'}</td>
<td>hypre</td>
<td>65.0</td>
<td>30.0</td>
</tr>
<tr>
<td>{&quot;name&quot;: 'mpi'}</td>
<td>mpi</td>
<td>35.0</td>
<td>20.0</td>
</tr>
<tr>
<td>{&quot;name&quot;: 'psm2'}</td>
<td>psm2</td>
<td>25.0</td>
<td>60.0</td>
</tr>
</tbody>
</table>
```
Pandas and dataframes

- Pandas is an open-source Python library for data analysis
- Dataframe: two-dimensional tabular data structure
  - Supports many operations borrowed from SQL databases

<table>
<thead>
<tr>
<th>Index</th>
<th>Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>node</td>
<td>name</td>
</tr>
<tr>
<td>0</td>
<td>{'name': 'main'}</td>
</tr>
<tr>
<td>1</td>
<td>{'name': 'physics'}</td>
</tr>
<tr>
<td>2</td>
<td>{'name': 'mpi'}</td>
</tr>
<tr>
<td>3</td>
<td>{'name': 'psm2'}</td>
</tr>
<tr>
<td>4</td>
<td>{'name': 'solvers'}</td>
</tr>
<tr>
<td>5</td>
<td>{'name': 'hype'}</td>
</tr>
<tr>
<td>6</td>
<td>{'name': 'mpi'}</td>
</tr>
<tr>
<td>7</td>
<td>{'name': 'psm2'}</td>
</tr>
</tbody>
</table>
Pandas and dataframes

- Pandas is an open-source Python library for data analysis

- Dataframe: two-dimensional tabular data structure
  - Supports many operations borrowed from SQL databases
  - MultiIndex enables working with high-dimensional data in a 2D data structure

<table>
<thead>
<tr>
<th>Rows</th>
<th>node</th>
<th>name</th>
<th>time (inc)</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>{'name': 'main'}</td>
<td>main</td>
<td>200.0</td>
<td>10.0</td>
</tr>
<tr>
<td>1</td>
<td>{'name': 'physics'}</td>
<td>physics</td>
<td>60.0</td>
<td>40.0</td>
</tr>
<tr>
<td>2</td>
<td>{'name': 'mpi'}</td>
<td>mpi</td>
<td>20.0</td>
<td>5.0</td>
</tr>
<tr>
<td>3</td>
<td>{'name': 'psm2'}</td>
<td>psm2</td>
<td>15.0</td>
<td>30.0</td>
</tr>
<tr>
<td>4</td>
<td>{'name': 'solvers'}</td>
<td>solvers</td>
<td>100.0</td>
<td>10.0</td>
</tr>
<tr>
<td>5</td>
<td>{'name': 'hype'}</td>
<td>hype</td>
<td>65.0</td>
<td>30.0</td>
</tr>
<tr>
<td>6</td>
<td>{'name': 'mpi'}</td>
<td>mpi</td>
<td>35.0</td>
<td>20.0</td>
</tr>
<tr>
<td>7</td>
<td>{'name': 'psm2'}</td>
<td>psm2</td>
<td>25.0</td>
<td>60.0</td>
</tr>
</tbody>
</table>
Central data structure: a *GraphFrame*

- Consists of a structured index graph object and a pandas dataframe
- Graph stores caller-callee relationships
- Dataframe stores all numerical and categorical data
Central data structure: a *GraphFrame*

- Consists of a structured index graph object and a pandas dataframe
- Graph stores caller-callee relationships
- Dataframe stores all numerical and categorical data
Central data structure: a GraphFrame

- Consists of a structured index graph object and a pandas dataframe
- Graph stores caller-callee relationships
- Dataframe stores all numerical and categorical data

![GraphFrame Diagram]

<table>
<thead>
<tr>
<th>node</th>
<th>name</th>
<th>nid</th>
<th>node</th>
<th>time</th>
<th>time (inc)</th>
</tr>
</thead>
<tbody>
<tr>
<td>main</td>
<td>main</td>
<td>0</td>
<td>main</td>
<td>40.0</td>
<td>200.0</td>
</tr>
<tr>
<td>physics</td>
<td>physics</td>
<td>1</td>
<td>physics</td>
<td>40.0</td>
<td>60.0</td>
</tr>
<tr>
<td>mpi</td>
<td>mpi</td>
<td>2</td>
<td>mpi</td>
<td>5.0</td>
<td>20.0</td>
</tr>
<tr>
<td>psm2</td>
<td>psm2</td>
<td>3</td>
<td>psm2</td>
<td>15.0</td>
<td>15.0</td>
</tr>
<tr>
<td>solvers</td>
<td>solvers</td>
<td>4</td>
<td>solvers</td>
<td>0.0</td>
<td>100.0</td>
</tr>
<tr>
<td>hypre</td>
<td>hypre</td>
<td>5</td>
<td>hypre</td>
<td>65.0</td>
<td>65.0</td>
</tr>
<tr>
<td>mpi</td>
<td>mpi</td>
<td>6</td>
<td>mpi</td>
<td>10.0</td>
<td>35.0</td>
</tr>
<tr>
<td>psm2</td>
<td>psm2</td>
<td>7</td>
<td>psm2</td>
<td>25.0</td>
<td>25.0</td>
</tr>
</tbody>
</table>
Dataframe operation: filter

\[
\text{filtered\_gf} = \text{gf.filter(}\lambda x: x['time'] > 10.0)\]

Dataframe operation: filter

```python
filtered_gf = gf.filter(lambda x: x['time'] > 10.0)
```

<table>
<thead>
<tr>
<th>node</th>
<th>name</th>
<th>nid</th>
<th>node</th>
<th>time</th>
<th>time (inc)</th>
</tr>
</thead>
<tbody>
<tr>
<td>main</td>
<td>main</td>
<td>0</td>
<td>main</td>
<td>40.0</td>
<td>200.0</td>
</tr>
<tr>
<td>physics</td>
<td>physics</td>
<td>1</td>
<td>physics</td>
<td>40.0</td>
<td>60.0</td>
</tr>
<tr>
<td>mpi</td>
<td>mpi</td>
<td>2</td>
<td>mpi</td>
<td>5.0</td>
<td>20.0</td>
</tr>
<tr>
<td>psm2</td>
<td>psm2</td>
<td>3</td>
<td>psm2</td>
<td>15.0</td>
<td>15.0</td>
</tr>
<tr>
<td>solvers</td>
<td>solvers</td>
<td>4</td>
<td>solvers</td>
<td>0.0</td>
<td>100.0</td>
</tr>
<tr>
<td>hypre</td>
<td>hypre</td>
<td>5</td>
<td>hypre</td>
<td>65.0</td>
<td>65.0</td>
</tr>
<tr>
<td>mpi</td>
<td>mpi</td>
<td>6</td>
<td>mpi</td>
<td>10.0</td>
<td>35.0</td>
</tr>
<tr>
<td>psm2</td>
<td>psm2</td>
<td>7</td>
<td>psm2</td>
<td>25.0</td>
<td>25.0</td>
</tr>
</tbody>
</table>
**Dataframe operation: filter**

```python
filtered_gf = gf.filter(lambda x: x['time'] > 10.0)
```

<table>
<thead>
<tr>
<th>node</th>
<th>name</th>
<th>nid</th>
<th>node</th>
<th>time</th>
<th>time (inc)</th>
</tr>
</thead>
<tbody>
<tr>
<td>main</td>
<td>main</td>
<td>0</td>
<td>main</td>
<td>40.0</td>
<td>200.0</td>
</tr>
<tr>
<td>physics</td>
<td>physics</td>
<td>1</td>
<td>physics</td>
<td>40.0</td>
<td>60.0</td>
</tr>
<tr>
<td>mpi</td>
<td>mpi</td>
<td>2</td>
<td>mpi</td>
<td>5.0</td>
<td>20.0</td>
</tr>
<tr>
<td>psm2</td>
<td>psm2</td>
<td>3</td>
<td>psm2</td>
<td>15.0</td>
<td>15.0</td>
</tr>
<tr>
<td>solvers</td>
<td>solvers</td>
<td>4</td>
<td>solvers</td>
<td>0.0</td>
<td>100.0</td>
</tr>
<tr>
<td>hypre</td>
<td>hypre</td>
<td>5</td>
<td>hypre</td>
<td>65.0</td>
<td>65.0</td>
</tr>
<tr>
<td>mpi</td>
<td>mpi</td>
<td>6</td>
<td>mpi</td>
<td>10.0</td>
<td>35.0</td>
</tr>
<tr>
<td>psm2</td>
<td>psm2</td>
<td>7</td>
<td>psm2</td>
<td>25.0</td>
<td>25.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>node</th>
<th>name</th>
<th>nid</th>
<th>node</th>
<th>time</th>
<th>time (inc)</th>
</tr>
</thead>
<tbody>
<tr>
<td>main</td>
<td>main</td>
<td>0</td>
<td>main</td>
<td>40.0</td>
<td>200.0</td>
</tr>
<tr>
<td>physics</td>
<td>physics</td>
<td>1</td>
<td>physics</td>
<td>40.0</td>
<td>60.0</td>
</tr>
<tr>
<td>psm2</td>
<td>psm2</td>
<td>3</td>
<td>psm2</td>
<td>15.0</td>
<td>15.0</td>
</tr>
<tr>
<td>solvers</td>
<td>solvers</td>
<td>4</td>
<td>solvers</td>
<td>0.0</td>
<td>100.0</td>
</tr>
<tr>
<td>hypre</td>
<td>hypre</td>
<td>5</td>
<td>hypre</td>
<td>65.0</td>
<td>65.0</td>
</tr>
<tr>
<td>psm2</td>
<td>psm2</td>
<td>7</td>
<td>psm2</td>
<td>25.0</td>
<td>25.0</td>
</tr>
</tbody>
</table>
We now describe some of the important operators provided by Hatchet. Figure 4 shows a dataframe before and after a filter operation on the dataframe. As shown in Figure 5, we consider a graph to be immutable, so any of the operations below are performed on the graphframe. As the readers construct pieces of information, the CCT structure is used to construct the graphframe object. As the users construct the graphframe, some only modify the dataframe, some only modify the graph, and others perform the aggregation over all MPI processes or threads at the per-node granularity. Additionally, a squash operation will make the values in all columns in the hierarchical index by specifying an aggregation function. Essentially, this performs a filter operation. When one or more columns containing inclusive metrics inaccurate, since the parent-child connections are modified, all the columns in the resulting dataframe that store inclusive values of some metric become inaccurate. This function allows the user to drop the additional index sections. Note that we consider a graph to be immutable, so any operations below are performed on the graphframe, some of the operations below are performed on the graphframe.

```python
filtered_gf = gf.filter(lambda x: x['time'] > 10.0)
```

<table>
<thead>
<tr>
<th>name</th>
<th>nid</th>
<th>node</th>
<th>time</th>
<th>time (incl)</th>
</tr>
</thead>
<tbody>
<tr>
<td>main</td>
<td>main</td>
<td>0</td>
<td>main</td>
<td>40.0</td>
</tr>
<tr>
<td>physics</td>
<td>physics</td>
<td>1</td>
<td>physics</td>
<td>40.0</td>
</tr>
<tr>
<td>mpi</td>
<td>mpi</td>
<td>2</td>
<td>mpi</td>
<td>5.0</td>
</tr>
<tr>
<td>psm2</td>
<td>psm2</td>
<td>3</td>
<td>psm2</td>
<td>15.0</td>
</tr>
<tr>
<td>solvers</td>
<td>solvers</td>
<td>4</td>
<td>solvers</td>
<td>0.0</td>
</tr>
<tr>
<td>hypre</td>
<td>hypre</td>
<td>5</td>
<td>hypre</td>
<td>65.0</td>
</tr>
<tr>
<td>mpi</td>
<td>mpi</td>
<td>6</td>
<td>mpi</td>
<td>10.0</td>
</tr>
<tr>
<td>psm2</td>
<td>psm2</td>
<td>7</td>
<td>psm2</td>
<td>25.0</td>
</tr>
</tbody>
</table>
Graph operation: squash

filtered_gf = gf.filter(lambda x: x['time'] > 10.0)
Graph operation: squash

filtered_gf = gf.filter(lambda x: x['time'] > 10.0)

squashed_gf = filtered_gf.squash()
Graph operation: squash

filtered_gf = gf.filter(lambda x: x['time'] > 10.0)

squashed_gf = filtered_gf.squash()
Graphframe operation: subtract

gf1 = ht.GraphFrame.from_literal( ... )
gf2 = ht.GraphFrame.from_literal( ... )
gf2 -= gf1
Graphframe operation: subtract

gf1 = ht.GraphFrame.from_literal( ... )
gf2 = ht.GraphFrame.from_literal( ... )
gf2 -= gf1
Graphframe operation: subtract

```
gf1 = ht.GraphFrame.from_literal(...)
gf2 = ht.GraphFrame.from_literal(...)
gf2 -= gf1
```
Graphframe operation: subtract

```
gf1 = ht.GraphFrame.from_literal( ... )
gf2 = ht.GraphFrame.from_literal( ... )
gf2 -= gf1
```

https://hatchet.readthedocs.io
Visualizing small graphs

```python
print(gf.tree(color=True))
```

0.000 foo
├ 5.000 bar
│ ├ 5.000 baz
│ │ 10.000 grault
├ 0.000 qux
│ ├ 5.000 quux
│ │ 10.000 corge
│ │ ├ 5.000 bar
│ │ │ ├ 5.000 baz
│ │ │ │ 10.000 grault
│ │ │ │ 10.000 grault
│ │ │ 15.000 garply
│ └ 0.000 waldo
│ ├ 5.000 fred
│ │ ├ 5.000 plugh
│ │ │ 5.000 xyzzy
│ │ │ └ 5.000 thud
│ │ │ └ 5.000 baz
│ │ │ │ 15.000 garply
│ │ └ 15.000 garply

5 PERFORMANCE
It is vital that performance analysis tools have low overheads and that they enable quick analysis of performance datasets without the user having to wait for a long time for each operation to complete. In Figure 9, we provide execution times for some operations in Hatchet when using increasingly large datasets. We ran LULESH...
Visualizing small graphs

```python
print(gf.tree(color=True))
```

```
0.000 foo
├── 5.000 bar
│   ├── 5.000 baz
│   │   ├── 10.000 grault
│   │   └── 10.000 grault
│   └── 15.000 garply
└── 0.000 waldo
    ├── 5.000 fred
    │   ├── 5.000 plugh
    │   │   ├── 5.000 thud
    │   │   └── 5.000 baz
    │   └── 15.000 garply
    └── 15.000 garply
```

with open("test.dot", "w") as dot_file:
    dot_file.write(gf.to_dot())
Visualizing small graphs

```python
print(gf.tree(color=True))
```

```
0.000 foo
├── 5.000 bar
│   ├── 5.000 baz
│   │   └── 10.000 grault
│   └── 0.000 gux
│     ├── 10.000 corge
│     │   └── 5.000 bar
│     │       ├── 5.000 baz
│     │       │   └── 10.000 grault
│     │       └── 10.000 grault
│     └── 15.000 garply
└── 0.000 waldo
    ├── 5.000 fred
    │   ├── 5.000 plugh
    │   │   └── 5.000 thud
    │   │       └── 5.000 baz
    │   └── 15.000 garply
    └── 15.000 garply
```

```python
with open("test.dot", "w") as dot_file:
    dot_file.write(gf.to_dot())
```

```python
with open("test.txt", "w") as folded_stack:
    folded_stack.write(gf.to_flamegraph())
```

Flamegraph
Example 1: Generating a flat profile

gf = ht.GraphFrame.from_hpctoolkit('kripke')
gf.drop_index_levels()

grouped = gf.dataframe.groupby('name').sum()
sorted_df = grouped.sort_values(by=['time'], ascending=False)
print(sorted_df)
Example 1: Generating a flat profile

gf = ht.GraphFrame.from_hpctoolkit('kripke')
gf.drop_index_levels()

grouped = gf.dataframe.groupby('name').sum()
sorted_df = grouped.sort_values(by=['time'], ascending=False)
print(sorted_df)
Example 1: Generating a flat profile

gf = ht.GraphFrame.from_hpctoolkit('kripke')
gf.drop_index_levels()

```python
grouped = gf.dataframe.groupby('name').sum()
sorted_df = grouped.sort_values(by=['time'], ascending=False)
```

print(sorted_df)
Example 2: Comparing two executions

gf1 = ht.GraphFrame.from_caliper('lulesh-1core.json')
gf2 = ht.GraphFrame.from_caliper('lulesh-27cores.json')

gf2.drop_index_levels()
gf3 = gf2 - gf1

sorted_df = gf3.dataframe.sort_values(by=['time'], ascending=False)
print(sorted_df)
Example 2: Comparing two executions

gf1 = ht.GraphFrame.from_caliper('lulesh-1core.json')
gf2 = ht.GraphFrame.from_caliper('lulesh-27cores.json')

gf2.drop_index_levels()
gf3 = gf2 - gf1

sorted_df = gf3.dataframe.sort_values(by=['time'], ascending=False)
print(sorted_df)
Example 2: Comparing two executions

gf1 = ht.GraphFrame.from_caliper('lulesh-1core.json')
gf2 = ht.GraphFrame.from_caliper('lulesh-27cores. json')

gf2.drop_index_levels()
gf3 = gf2 - gf1

sorted_df = gf3.dataframe.sort_values(by=['time'], ascending=False)

print(sorted_df)

<table>
<thead>
<tr>
<th>node</th>
<th>name</th>
<th>nid</th>
<th>time</th>
<th>time (inc)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TimIncrement</td>
<td>TimIncrement</td>
<td>25.0</td>
<td>8.505048e+06</td>
<td>8.505048e+06</td>
</tr>
<tr>
<td>CalcQForElems</td>
<td>CalcQForElems</td>
<td>16.0</td>
<td>4.455672e+06</td>
<td>5.189453e+06</td>
</tr>
<tr>
<td>CalcHourglassControlForElems</td>
<td>CalcHourglassControlForElems</td>
<td>7.0</td>
<td>3.888798e+06</td>
<td>4.755817e+06</td>
</tr>
<tr>
<td>LagrangeNodal</td>
<td>LagrangeNodal</td>
<td>3.0</td>
<td>1.986046e+06</td>
<td>8.828475e+06</td>
</tr>
<tr>
<td>CalcForceForNodes</td>
<td>CalcForceForNodes</td>
<td>4.0</td>
<td>1.017857e+06</td>
<td>6.842429e+06</td>
</tr>
</tbody>
</table>
Example 3: Scaling study

datasets = glob.glob('lulesh*.json')
datasets.sort()

dataframes = []
for dataset in datasets:
gf = ht.GraphFrame.from_caliper(dataset)
gf.drop_index_levels()

    num_pes = re.match('(.*)-(\d+)(\.*', dataset).group(2)
gf.dataframe['pes'] = num_pes
filtered_gf = gf.filter(lambda x: x['time'] > 1e6)
dataframes.append(filtered_gf.dataframe)

result = pd.concat(dataframes)
pivot_df = result.pivot(index='pes', columns='name', values='time')
pivot_df.loc[:,:,:].plot.bar(stacked=True, figsize=(10,7))
Example 3: Scaling study

datasets = glob.glob('lulesh*.json')
datasets.sort()

dataframes = []
for dataset in datasets:
    gf = ht.GraphFrame.from_caliper(dataset)
    gf.drop_index_levels()
    
    num_pes = re.match('(.*)-(\d+)(.*)', dataset).group(2)
    gf.dataframe['pes'] = num_pes
    filtered_gf = gf.filter(lambda x: x['time'] > 1e6)
    dataframes.append(filtered_gf.dataframe)

result = pd.concat(dataframes)
pivot_df = result.pivot(index='pes', columns='name', values='time')
pivot_df.loc[::,:].plot.bar(stacked=True, figsize=(10,7))
Example 3: Scaling study

datasets = glob.glob('lulesh*.json')
datasets.sort()

dataframes = []
for dataset in datasets:
gf = ht.GraphFrame.from_caliper(dataset)
gf.drop_index_levels()  
num_pes = re.match('(.*)-\d+(.*ynth)', dataset)
gf.dataframe['pes'] = num_pes
filtered_gf = gf.filter(lambda x: x['name'] == 'MPI_Send')
dataframes.append(filtered_gf.dataframe)

result = pd.concat(dataframes)
pivot_df = result.pivot(index='pes', columns='name', values='time')
pivot_df.loc[::, :].plot.bar(stacked=True, figsize=(10, 7))
Abhinav Bhathele
5218 Brendan Iribe Center (IRB) / College Park, MD 20742
phone: 301.405.4507 / e-mail: bhatele@cs.umd.edu