Useful stuff

Amdahl's Law $speedup = \frac{1}{\frac{f}{s}+(1-f)}$

Erlang Cheat Sheet

lists:foldl

```
fsum(L) -> lists:foldl(fun(X,S) -> S+X end, 0, L).
```

reduce

```
largest_gap_seq([]) -> {};
largest_gap_seq([V]) -> {V, V};
largest_gap_seq([V1, V2 | Tail]) ->
    F = fun(XNew, {CurGap, XOld}) -> {gap_max(CurGap, {XOld, XNew}), XNew} end,
    {Gap, _XLast} = lists:foldl(F, {{V1, V2}, V2}, Tail),
    Gap.

largest_gap_par(WTree, DataKey) ->
  Leaf = fun(ProcState) -> leaf(workers:get(ProcState, DataKey)) end,
  Combine = fun(Left, Right) -> combine(Left, Right) end,
  Root = fun(Value) -> root(Value) end,
  wtree:reduce(WTree, Leaf, Combine, Root).

leaf([]) -> {};
leaf([V]) -> {V, {V,V}, V};
leaf(List) -> {hd(List), largest_gap_seq(List), lists:last(List)}.

combine({}, Right) -> Right;
combine(Left, {}) -> Left;
combine({L1, {L2, L3}, L4}, {R1, {R2, R3}, R4}) ->
  {L1, gap_max(gap_max({L2, L3}, {L4, R1}), {R2, R3}), R4}.

root({_MinV, {V1, V2}, _MaxV}) -> {V1, V2}.

gap_max({A1, A2}, {B1, B2}) when (A2 - A1) >= (B2 - B1) -> {A1, A2};
gap_max(_, B) -> B.
```

scan

```
bank_statement(W, SrcKey, DstKey, InitialBalance) ->
  Leaf1 = fun(ProcState) ->
    process_transactions(wtree:get(ProcState, SrcKey), {1, 0})
  end,
  Leaf2 = fun(ProcState, AccIn) ->
    {_S, C} = AccIn,
    Src = wtree:get(ProcState, SrcKey),
    Result = process_transactions_cum(Src, C),     % compute the cummulative sum
    wtree:put(ProcState, DstKey, Result) % save the result -- must be the last expression
  end,                                   %   in the Leaf2 function
  Combine = fun({C1, S1}, {C2, S2}) -> {C1*C2, S1*C2 + S2} end,
  wtree:scan(W, Leaf1, Leaf2, Combine, {1, InitialBalance}).
```

cuda

```
__shared__ float v[1024];
__device__ float f(float x) {
  return ((5/2)*(x*x*x - x));
}

__device__ void compute and reduce(uint n, uint m, float *x) {
  uint myId = threadIdx.x;
  if(myId < n) {
    float y = x[myId];
    for(uint i = 0; i < m; i++) {
      y = f(y); % y = f^(i + 1) (x[myId])
    }
    v[myId] = y; % shared memory is much faster than global memory
    for(uint m = n >> 1; m > 0; m = n >> 1) { % reduce
      n -= m;
      __syncthreads();
      if(myId < m) {
    v[myId] += v[myId+n];
      }
    }
    x[myId] = v[myId]; % move result to global memory
  }
}
```