Bitonic Sort: Parallel Analysis

Toby Isaac

CSE 6230, Fall 2017

1 Pseudocode

Algorithm 1: `sortBitonic`(r, P, n, A, d, b)

Data: r is the process's rank;
P is the number of processes (power of 2);
n is the number of local keys (same on all processes);
A is the array of keys;
d is `True` if sorting forward, `False` if sorting backward;
b is `True` if A is bitonic on input.

```
if P = 1 then
    if b then
        /* local sort when input is bitonic, O(n log_2 n)            */
        sortLocalBitonic(n, A, d)
    else
        /* local sort with no special condition, O(n log_2 n)        */
        /* (but we expect slower than sortLocalBitonic())            */
        sortLocal(n, A, d)
else
    if not b then
        /* Make A bitonic, sort half forward, half backward          */
        /* ⊗ is exclusive or                                         */
        sortBitonic(r mod P/2, P/2, n, A, d ⊗ (r ≥ P/2), False)
    /* Swap splits keys into lesser and greater halves               */
    /* (or greater and lesser if backward direction)                 */
    swapBitonic(r, P, n, A, d)
    /* Recursive sort of distributed bitonic sequence                */
    sortBitonic(r mod P/2, P/2, n, A, d, True)
```

Algorithm 2: `swapBitonic`(r, P, n, A, d)

Data: r is the process's rank;
P is the number of processes (power of 2);
n is the number of local keys (same on all processes);
A is the array of keys;
d is `True` if sorting forward, `False` if sorting backward.

```
   /* Communicating partner in the other half */
1  q ← r + P/2 mod P
2  A_recv ← Sendrecv(A, n, q)
   /* ⊗ is exclusive or */
3  if (r < q) ⊗ d then
       /* Swap to get lesser value */
4      for 0 ≤ i < n do
5          A[i] ← min{A[i], A_recv[i]}
6  else
       /* Swap to get greater value */
7      for 0 ≤ i < n do
8          A[i] ← max{A[i], A_recv[i]}
```

2 Runtime analysis

We want an expression for $T_b(n, p)$, the runtime of parallel bitonic sort in terms of n, the number of local keys, and P, the number of processes.

We will first get $\tilde{T}_b(n, p)$, the runtime of parallel sort when the input is bitonic (b is `True`). In this case

$$\tilde{T}_b(n, 1) = T_{lb}(n) := T_{\texttt{sortLocalBitonic}}(n) \in O(n \log_2 n).$$

Then, the recursive case:

$$\tilde{T}_b(n, P) = (T_{sb}(n, P) := T_{\texttt{swapBitonic}}(n, P)) + \tilde{T}_b(n, P/2).$$

We will assume that network congestion is not a problem so that $T_{sb}(n, P)$ is independent of P, $T_{sb}(n)$. Ignoring for a moment the cost of local operations in `swapBitonic` (the loops at lines 4 and 7), the cost is at least the cost of a `Sendrecv` of n keys. In the parameters of the `LogGP` model, the cost of a `Sendrecv` should be

$$T_{sb}(n) \geq L + 2o + G(kn - 1),$$

where L is the network latency, o is the overhead of starting or finishing a communication, G is the inverse of the network bandwidth (i.e., G has units of s/B), and k is the number of bytes per key (8, in our case).

Expanding the recursion, we get

$$\begin{aligned}\tilde{T}_b(n, P) &= \underbrace{T_{sb}(n) + \cdots + T_{sb}(n)}_{\times \log_2 P} + \tilde{T}_b(n, 1) \\ &= T_{sb}(n) \log_2 P + T_{lb}(n).\end{aligned}$$

Now we solve for the full runtime $T_b(n,P)$ when the inputs are not bitonic (b is `False`), the default case.

The base case:

$$T_b(n,1) = T_l(n) := T_{\texttt{sortLocal}}(n) \in O(n \log_2 n).$$

The recursive case:

$$\begin{aligned} T_b(n,P) &= T_b(n,P/2) + \tilde{T}_b(n,P) \\ &= \underbrace{\tilde{T}_b(n,P) + \tilde{T}_b(n,P/2) + \cdots + \tilde{T}_b(n,2)}_{\times \log_2 P} + T_b(n,1). \end{aligned}$$

We insert our expression for $\tilde{T}_b(n,P)$, group like terms, and use the property of logarithms to turn $\log_2 P/K$ into $\log_2 P - \log_2 K$:

$$\begin{aligned} T_b(n,P) &= T_{sb}(n)(\log_2 P + \log_2 P/2 + \cdots + \log_2 2) + \\ &\quad T_{lb}(n)(\underbrace{1+1+\cdots+1}_{\times \log_2 P}) + T_b(n,1) \\ &= T_{sb}(n)(\underbrace{\log_2 P + \log_2 P + \cdots + \log_2 P}_{\times \log_2 P}) - \\ &\quad T_{sb}(n)(0+1+\cdots+(\log_2 P - 1)) + \\ &\quad T_{lb}(n)(\underbrace{1+1+\cdots+1}_{\times \log_2 P}) + T_b(n,1) \\ &= T_{sb}(n)\left(\log_2^2 P - \sum_{i=1}^{\log_2 P - 1} i\right) + T_{lb}(n)\log_2 P + T_l(n) \\ &= T_{sb}(n)(\log_2^2 P - \tfrac{1}{2}(\log_2 P - 1)\log_2 P) + T_{lb}(n)\log_2 P + T_l(n) \\ &= \tfrac{1}{2}T_{sb}(n)(\log_2^2 P + \log_2 P) + T_{lb}(n)\log_2 P + T_l(n). \end{aligned}$$

We now have a runtime expression for the full `sortBitonic`() function in terms of its components that are simpler to analyze/measure on their own: `sortLocal`(), `sortLocalBitonic`(), and `swapBitonic`(), which we take to be $O(n \log n)$, $O(n \log n)$ and $O(n)$, respectively. If we trust our models of the runtimes of those components, then our model lets us:

- Predict the performance of `sortBitonic`().
- Verify and validate the performance of `sortBitonic`(). If our measured times do not match the model, the implementation is wrong or the model is wrong (e.g., our pseudocode omits something critical).
- Speculatively compare the performance of other sort implementations to `sortBitonic`().

If we *don't* already have good models of the component algorithms, we can use the functional form of $T_b(n,P)$ to define an empirical model.

3 Empirical Model

First, let us approximate the components with some as yet undetermined coefficients:

$$T_{sb}(n) \approx C_{sb}^0 + C_{sb}^1 \; n,$$
$$T_{lb}(n) \approx C_{lb} \; n \log_2 n,$$
$$T_l(n) \approx C_l \; n \log_2 n.$$

[We include the constant term C_{sb}^0 because we think network latency may be significant in the case of small n.]

Expanding $T_b(n, P)$:

$$T_b(n, P) \approx \tfrac{1}{2}(C_{sb}^0 + C_{sb}^1 n)(\log_2^2 P + \log_2 P) + C_{lb} \; n \log_2 n \log_2 P + C_l \; n \log_2 n.$$

Now we could measure the runtime of `sortBitonic`(n, P) for a large number of (n, P) pairs, and do least-squares fit for the coefficients, and use a statistical measure of goodness of fit to decide if our four-coefficient model is useful.

We can also isolate some of the coefficient with subsets of the parameter space. If $P = 1$, then C_l (the coefficient for the local sort) is the only one that has an effect, so we can estimate C_l in isolation. If $N = 1$, then we can estimate $(C_{sb}^0 + C_{sb}^1)$ (the minimum cost of performing `swapBitonic`()).