Jakub Šoustar, xsoust02

Minimum Extraction Sort

Rozbor

Algoritmus vytváří seřazenou sekvenci hodnot pomocí perfektně vyváženého binárního stromu, jehož uzly tvoří jednotlivé procesory. Z tohoto důvodu je algoritmus schopný pracovat pouze s posloupnostmi o délce 2^k pro nějaké $k \geq 0$. Strom je inicializován načtením hodnot do listových uzlů (jedna hodnota na každý procesor). Samotné řazení probíhá paralelním extrahováním nejmenší, případně největší, hodnoty do kořene každého podstromu. Hodnoty, které takto doputují až ke kořenovému procesoru celého stromu, jsou tímto postupně předávány na výstup.

Protože algoritmus pracuje s perfektně vyváženým binárním stromem, je pro sekvenci n hodnot zapotřebí $p = 2 \cdot n - 1$ procesorů (n listových a $n-1$ nelistových). Pro zhodnocení časové složitosti budeme uvažovat kořenový procesor. Tento procesor musí postupně extrahovat a vypsat n hodnot, celkem tedy $2 \cdot n$. První hodnota z listového procesoru doputuje do kořenové procesoru po $\log_2(2)+1$ krocích a každá další po dvou krocích. Z toho získáváme tyto teoretické složitosti:

$$t(n) = 2 \cdot n + \log_2(n) - 1 = O(n)$$

$$p(n) = 2 \cdot n - 1 = O(n)$$

$$c(n) = t(n) \cdot p(n) = 0(n^2)$$

Implementace

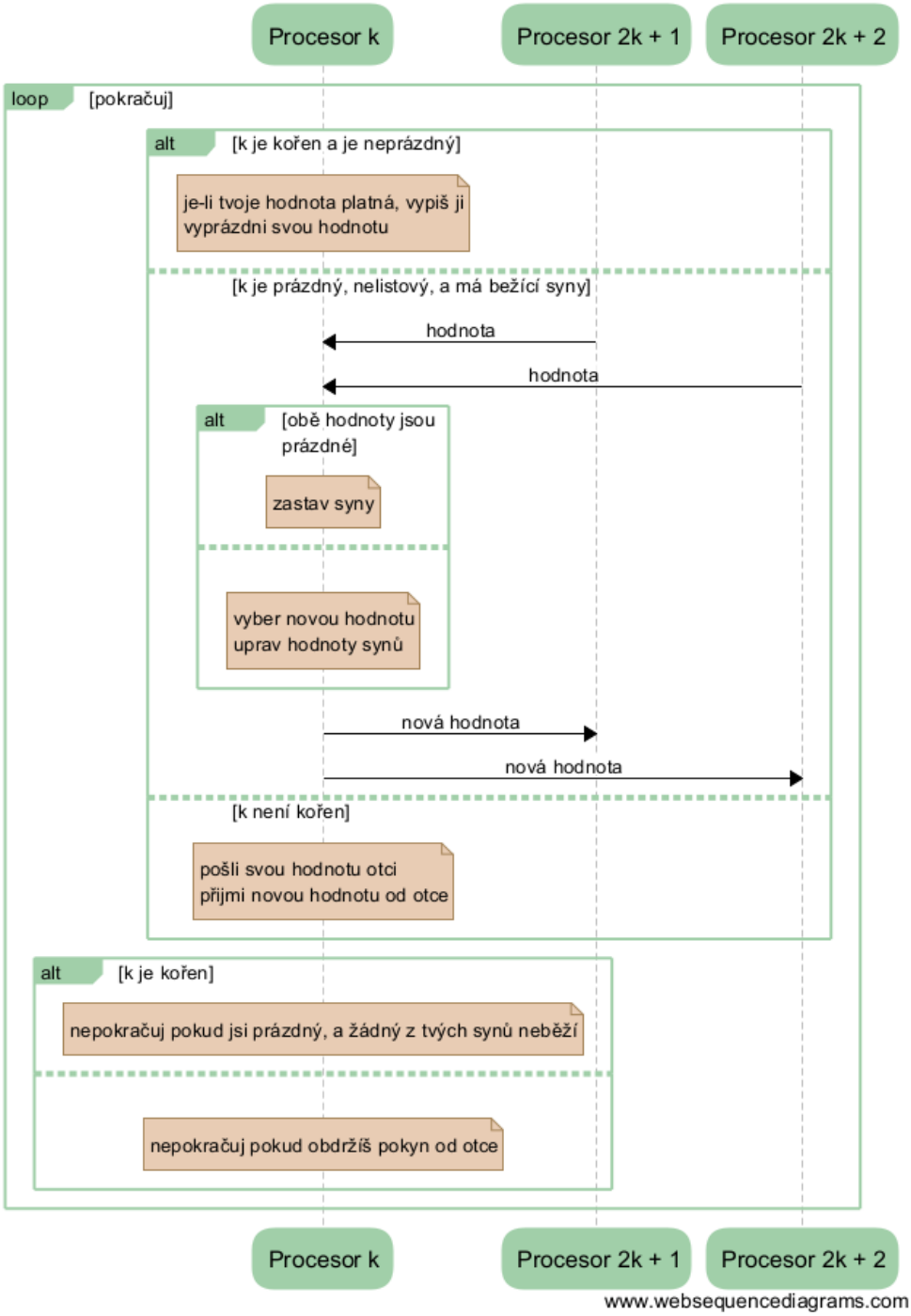
Algoritmus je implementován následujícím způsobem:

1. Kořenový procesor načte vstupní soubor. Je-li potřeba, sekvence je doplněna speciální hodnotou tak, aby celkový počet hodnot byl mocninou dvou. Následně jsou hodnoty po jedné odeslány všem listovým procesorům. Listové procesory jsou naplněny získanou hodnotou. Ostatní uzly se inicializují prázdnou hodnotu. V krajním případě řazení jediné hodnoty kořenový procesor pouze hodnotu vypíše a algoritmus se ukončí.
2. Po inicializační fázi se paralelně v cyklu provádí:
   a) Každý procesor provede právě jednu z následujících operací:
      i. Je-li uzel kořenem a je neprázdný, pak vypíše svoji hodnotu. Výpis se neuskuteční jde-li o výplňovou hodnotu.
      ii. Je-li uzel nelistový, prázdný a oba jeho synové běží, pak přijme hodnoty svých synovských uzlů. Uzel přejme menší z obou hodnot za svou. Synovskému uzlu, jehož hodnotu uzel přejal, je zpět odeslána prázdná hodnota. Druhému uzlu je odeslána nezměněná hodnota. Je-li jedna z hodnot ze synovských uzlů prázdná, je vybrána ta druhá. Jsou-li obě prázdné, uzel se stává, nebo setrvává prázdný, a oběma synům je odeslána hodnota signalizující pokyn k ukončení.
      iii. Pokud uzel není kořenem pak odešle svoji hodnotu svému otci a čeká, až mu otec zašle zpět jeho novou hodnotu.
   b) Je-li uzel kořen, je prázdný a oba jeho synové byli zastaveni, dojde k ukončení provádění cyklu. Ostatní uzly ukončují provádění cyklu na příkaz otcovského procesoru.

Jakub Šoustar, xsoust02

Komunikační protokol

Na následujícím diagramu je znázorněn komunikační protokol z pohledu procesoru s rankem k. Tento procesor, není-li listový, má dva synovské procesory s ranky $2 \cdot k + 1$ a $2 \cdot k + 2$. Tyto tři procesory dohromady tvoří strom, jehož kořenový uzel může být zároveň jedním ze synů jiného procesoru a oba synové mohou být kořeny dalších stromů. Z tohoto důvodu je komunikace ilustrována právě z pohledu kořene tohoto jednoduchého stromu. Na základě tohoto protokolu probíhá komunikace v libovolně větším stromu.

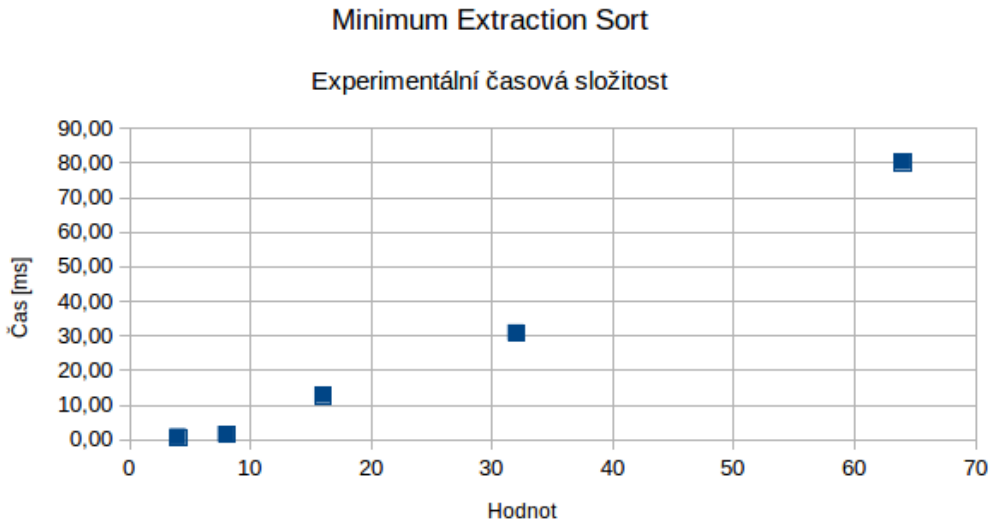


Jakub Šoustar, xsoust02

Experimenty

Časová složitost byla měřena pro náhodně generované hodnoty v počtech 4, 8, 16, 32 a 64. Pro každý počet hodnot bylo provedeno celkem 100 měření. V grafu jsou zaneseny průměry získaných hodnot. Měření bylo provedeno na osobním počítači, protože na školním serveru nebylo možné provést měření pro více jak 8 hodnot. Měření řazení dvojic nebylo zahrnuto, protože pro takovou úlohu není použití paralelismu zcela smysluplné. Pro 128 a více hodnot se mi již nepovedlo hodnoty naměřit i na mém počítači.

Naměřené hodnoty představují dobu běhu kořenového uzlu. Byla měřena pouze doba běhu hlavní smyčky algoritmu. V průběhu měření byly z kódu pomocí direktiv preprocesoru odstraněny veškeré výpisy na standardní výstup, s výjimkou naměřeného času, který je vypisován na standardní chybový výstup. Měření se provádí s použitím funkcí `MPI_Barrier` a `MPI_Wtime` z knihovny Open MPI.



Závěr

Experimentálně naměřená složitost se blíží té teoretické. Pokud uvážíme případné řazení větších souborů hodnot, půjde spíše o složitost $n \cdot \log(n)$ či horší. Pro potvrzení by bylo vhodné provést další měření pro více hodnot v prostředí, které to umožní. Odchylky od teoretické složitosti nejsou až tak překvapivé. Jakákoliv implementace algoritmu bude vždy navíc ovlivněna režií použité knihovny, hardware na kterém program poběží (množství virtuálních procesorů na fyzický a s tím spojená režie pro přepínání kontextů atp.), operačním systémem, který plánuje běh jednotlivých procesorů a dalšími okolnostmi. S rostoucím počtem procesorů také poroste režie na jejich synchronizaci a vzájemné zasílání zpráv.

Nejnáročnější částí algoritmu je jeho začátek, kdy běží největší počet procesorů. Protože ze začátku nejsou vnitřní uzly stromu zaplněny, dochází v každém kroku algoritmu k postupně utichající komunikační explozi, jejíž náročnost na vyřízení může, v závislosti na parametrech běhového prostředí, růst strměji než lineárně (v závislosti na počtu procesorů). Také se dá předpokládat, že program pro jisté vstupní parametry již nebude schopen běhu či dokonce spuštění. Teoretický algoritmus tyto okolnosti nezohledňuje.

Na závěr také stojí za to uvést, že tento algoritmus má několik nedostatků, které je možné bez větších zásahů odstranit, a tak i celý algoritmus potencionálně optimalizovat. Prvním problémem je využití listových procesorů jakožto skladu pro jedinou hodnotu. Pokud bychom umístili hodnoty po dvou na jeden procesor, dosáhneme zmenšení počtu potřebných procesorů o polovinu. Druhým nedostatkem je nadbytečné užívání synovských procesorů. Návrhem sofistikovanějšího řízení komunikace by šlo docílit toho, aby se procesory ukončily okamžitě jakmile již nebudou třeba.