Usporedba sekvencijalnih i paralelnih algoritama za sortiranje

1st Robert Kunštek
Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva
Zagreb, Hrvatska
robert.kunstek@fer.hr

2nd Marko Rasonja
Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva
Zagreb, Hrvatska
marko.rasonja@fer.hr

3rd Noa Margeta
Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva
Zagreb, Hrvatska
noa.margeta@fer.hr

***Abstract*—This document is a model and instructions for LaTeX. This and the IEEEtran.cls file define the components of your paper [title, text, heads, etc.]. *CRITICAL: Do Not Use Symbols, Special Characters, Footnotes, or Math in Paper Title or Abstract.**

***Index Terms*—paralelizam, paralelno procesiranje, sortiranje, višedretveno procesiranje**

I. Uvod

Ideja projekta, kao što naslov kaže, je usporediti vremena izvođenja sekvencijalnih i paralelnih algoritama. Neki od algoritama koje ćemo obraditi su *mergesort, bubblesort, quicksort i timsort*. Uz usporedbu vremena izvođenja provest ćemo i teorijsku analizu vremenske složenosti algoritama. Za programsku izvedbu algoritama koristili smo jezik Chapel.

II. Background

A. Korištena tehnologija

Chapel je programski jezik visoke razine dizajniran za jednostavnije pisanje efikasnih paralelnih programa. U Chapelu su ugrađeni tipovi podataka i metode spremne za rad u više dretvi/procesa. Sadrži i paralelne petlje te brojne procedure za sinkronizaciju paralelnih zadataka. Ova obilježja učinila su ga savršenim kandidatom i konačno našim izborom za ovaj projekt.

B. Slični radovi (eng. Related work)

Postoji puno istraživanja u području povezanim sa našim radom. Većina se bavi dodatnom optimizacijom postojećih paralelnih algoritama ili izradom nekih potpuno novih algoritama.

T. Dobravec i D. Božidar usporedili su 7 paralelnih i sekvencijskih algoritama, sekvencijske u C++-u, a paralelne na grafičkim karticama preko platforme CUDA [1]. Vršili su usporedbu na više različitih oblika podataka (32-bitni brojevi, 64-bitni brojevi, ...) iz nekoliko različitih razdioba (gausova, uniformna, silazno sortirana, ...) Pokazali su kako je radix sort najbrži među sekvencijskim algoritmima osim za podatke iz sortirane distribuciju. Za paralelne algoritme merge sort bio je brži za velike podatke od radixa.

D. Zurek i drugi uspoređivali su nekoliko paralelnih algoritama sa svojim sekvencijskim pandanima i koristili višejezgrene procesore i grafičke kartice [2]. Pokazali su da su algoritmi izvedeni na više jezgri efikasniji i bolje se skaliraju. Algoritmi koji su se izvršavali i na grafičkoj kartici i na procesoru bili su brži od onih koji su se izvršavali samo na grafičkoj kartici, a najbrži bio je hibridni paralelni merge/quicksort algoritam izvršen samo na procesoru.

Rad K. Sujatha i drugih, pravio je usporedbu performansi algoritama za traženje pri izvođenju na jednojezgrenim i višejezgrenim procesorima [3]. Pokazali su da je program znatno efikasniji u potrošnji energije i vremenu izvođenja na višejezgrenim procesorima.

III. Rezultati

Samo testiranje algoritama vršeno je na računalu s operacijskim sustavom Linux Ubuntu 22.04.1 s 6 raspoloživih dretvi te 3GB RAM memorije. Svaki od algoritama testiran je uz korištenje podataka prikladne velicine te za svaku od mogućnosti raspoloživih dretvi. Količina podataka je varirala, no za svaki test korištena su 3 različita reda veličina kako bismo dobili rezultate i na manjem i na većem broju podataka. Testovi su ponavljani 5 puta te smo kao konačnu vrijednost uzimali srednju vrijednost dobivenih mjerenja. Za outliere u setu mjerenja su vršena dodatna mjerenja kako bi se dobila jasnija srednja vrijednost.

A. Rezultati sekvencijalnog izvođenja

Kao prvi primjer u našem projektu uzeli smo "Radix sort", algoritam sortiranja koji je među bržim sekvencijalnim algoritmima.

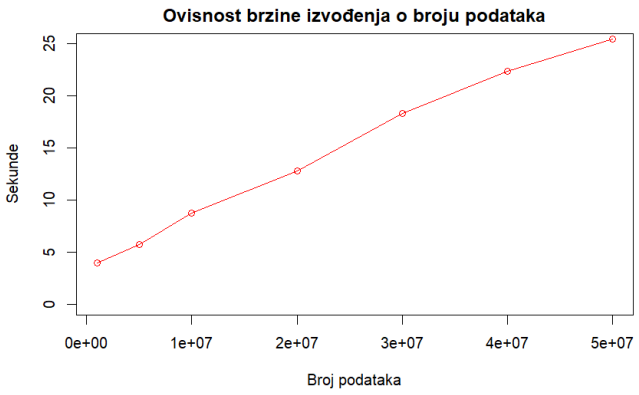

Figure: Prikaz performansi sekvencijalnog radix sorta

Kao što prikazuju rezultati, porast vremenske složenosti je blizu linearnog. Vremenska složenost Radix sorta je O(nk), gdje je *n* broj podataka, a *k* broj znamenki najvećeg broja.

B. Uvodno paralelno izvođenje

Nakon prikaza sekvencijalnog izvođenja, napravili smo testove za "even-odd" verziju paralelnog sorta. Rezultati testa su prikazani na slici.

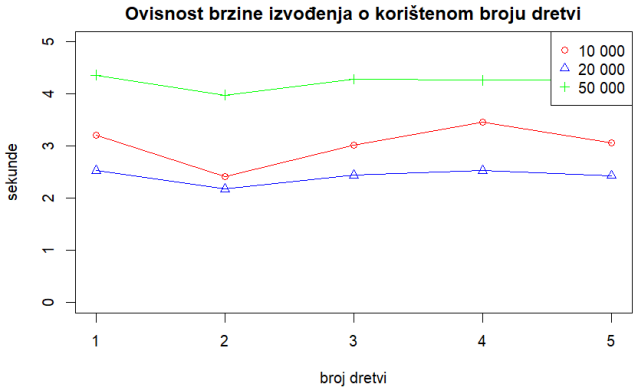

Figure: Prikaz performansi bubble sorta

Moramo napomenuti da je za prikaz rezultata korištena logaritamska skala na y-osi, osim za rezulatate dobivene koristeći 10 000 podataka iz razloga što su vremenska izvođenja tih mjerenja bila manja od 1 sekunde te bi u logaritamskoj skali bila negativna. Rezultati su očekivani iz razloga što naša verzija implementacije paralelnog bubble sorta koristi princip podjele posla na parne i neparne pozicije liste za sortiranje. Zbog toga se jedino poboljšanje u odnosu na sekvencijalno izvođenje očituje kod korištenja 2 dretve. U tom slučaju, tako implementirani bubble sort algoritam postiže najbolje rezultate, jer jedna dretva vrši zamjenu elemenata parnih pozicija, dok druga neparnih pozicija. Zamjena elemenata vrši se po principu bubble sorta.

C. Paralelno izvođenje

Nakon razmatranja sekvencijalnog radix sorta te djelomično paralelnog bubble sorta, testirali smo sljedeće algoritme: Merge sort, Quick sort i Tim sort. Svi algoritmi se temelje na principu podijeli-pa-vladaj.

1) Merge Sort: Merge sort je algoritam sortiranja asimptotske vremenske složenosti $O(n \log n)$. Korištenjem većeg broja dretvi, složenost se može smanjiti za konstantan broj puta. Rezultati su prikazani na slici.

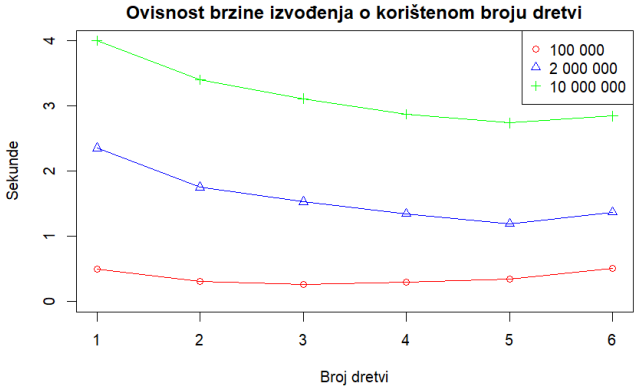

Figure: Prikaz performansi Merge sorta

Rezultati su kao i na prethodnim slikama, prikazani na logaritamskoj skali y-osi. Možemo uočiti da na manjem broju podataka (100 000), nema poboljšanja vremena izvođenja, no iz rezultata većeg broja podataka, vidljivo je poboljšanje u odnosu na sekvencijalno izvođenje. Tijekom testiranja, pokazalo se da je optimalan broj dretvi 5. Tada je poboljšanje:

$$S = \frac{T_{sequential}}{T_{parallel}} = \frac{54.9}{15.56} = 3.53 \quad (1)$$

Za broj dretvi veći od 5, "overhead" mijenjanja konteksta, upravljanje većim brojem dretvi te prevelika podjela podataka na manje dijelove u našoj implementaciji utječu na degradiranje performansi.

2) Tim Sort: Timsort je hibridni, stabilni algoritam sortiranja, izvedenica insertion sorta i merge sorta, dizajniran za dobru izvedbu na mnogim vrstama podataka. Asimptotska vremenska složenost merge sorta je $O(n \log n)$. Dobiveni rezultati prikazani su na sljedećoj slici u logaritamskoj skali na y-osi.

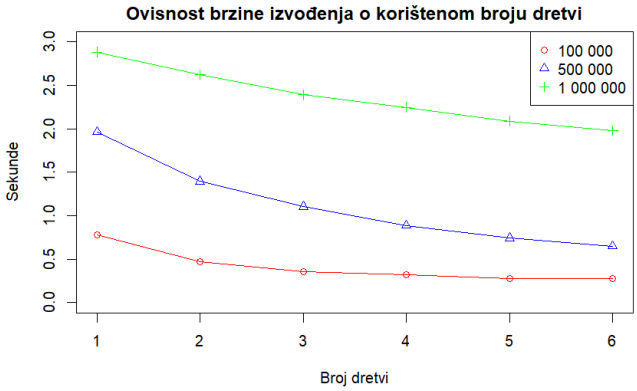

Figure: Prikaz performansi Tim sorta

Pregledom rezultata zaključili smo da se povećanjem broja dretvi izvođenje programa ubrzava. Zbog ograničenja u hardware-u, nismo bili u mogućnosti isprobati na većem

broju dretvi od 6. Poboljšanje iznosi:

$$S = \frac{T_{sequential}}{T_{parallel}} = \frac{17.894}{7.242} = 2.47 \quad (2)$$

Za razliku od Merge sorta, tim sort daje manje ubrzanje, no kao što smo napomenuli, zbog ograničenja u hardware-u, nismo bili u mogućnosti dobiti veća ubrzanja Tim sorta. Također, potrebno je napomenuti da su algoritmi testirani na različitom broju podataka te iz tog razloga usporedba nije u potpunosti definirana.

3) Quick sort: Quick sort je poslijednji algoritam sortiranja koji smo testirali. Njegova asimptotska složenost je $O(n^2)$ jer ovisi o broju pivota. Prosječna vremenska složenost iznosi $O(n \log n)$. Rezultati naših testiranja prikazani su na sljedećoj slici sa logaritamskoj skalom y-osi.

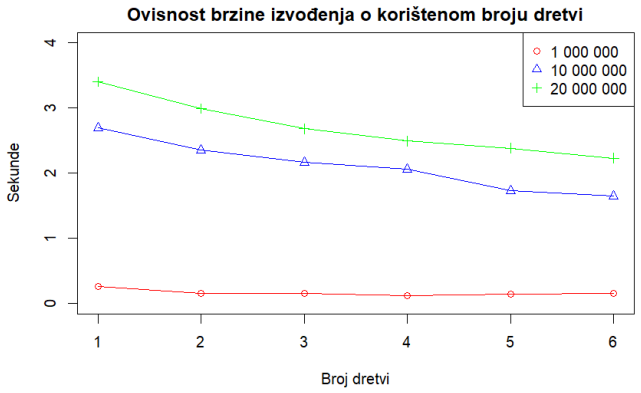

Figure: Prikaz performansi Quick sorta

Kao i kod Tim sorta, nismo bili u mogućnosti vidjeti granice poboljšanja uvođenjem paralelizma, no vidimo da se povećanjem dretvi smanjuje vrijeme izvođenja. Quick sort je pokazao relativno najbolje rezulate u izvođenju. Relativno iz razloga što nisu svi algoritmi ispitivani na jednakim brojevima podataka. Poboljšanje za 20 milijuna podataka koje smo dobili iznosi:

$$S = \frac{T_{sequential}}{T_{parallel}} = \frac{30.613}{9.311} = 3.29 \quad (3)$$

Iz poboljšanja je također vidljivo kako je Quick sort pokazao najbolje rezultate. Merge sort je imao bolje poboljšanje, no njegovo poboljšanje nije bilo konstantno s povećanjem broja dretvi, odnosno bilo je najveće uz korištenje 5 dretvi. U procesu implementacije korištena su 2 različite metode biranja pivota, no najboljom se pokazala metoda biranja pivota između prvog, srednjeg i zadnjeg člana liste za sortiranje.

ZAKLJUČCI

Zaključno, projekt usporedbe sekvencijalnih i paralelnih algoritama za sortiranje imao je za cilj istražiti učinkovitost različitih paralelnih metoda sortiranja za velike skupove podataka. Tijekom ovog projekta isprobali smo nekoliko različitih algoritama sortiranja. Neke nismo bili u mogućnosti paralelizirati ili smo ih djelomično paralelizirali.
Algoritmi s najboljim poboljšanjima u ovisnosti o broju dretvi su bili algoritmi koji se temelje na principu podijeli-pa-vladaj. Algoritmi za paralelno sortiranje koji koriste načelo podijeli-pa-vladaj smatraju se među najboljima jer iskorištavaju mogućnosti paralelne obrade modernih računalnih arhitektura. Oni funkcioniraju razlažući veliki problem na manje potprobleme koji se mogu samostalno riješiti, a zatim kombinirati kako bi se pronašlo rješenje izvornog problema.
U kontekstu našeg istraživanja, to znači da se skup podataka najprije podijeli na manje podnizove, koji se zatim paralelno neovisno sortiraju. Ovi podnizovi se zatim kombiniraju kako bi formirali konačni sortirani niz. Ovaj pristup omogućuje učinkovito korištenje više procesora ili jezgri, što može značajno smanjiti vrijeme potrebno za sortiranje velikih skupova podataka.

BUDUĆI RAD

Buduće mogućnosti rada za projekt algoritama paralelnog sortiranja uključuju daljnju optimizaciju algoritama za paralelno sortiranje. Još uvijek ima prostora za poboljšanje u smislu učinkovitosti i performansi algoritama za paralelno sortiranje implementiranih u ovom projektu. Također, možemo istražiti nove tehnike paralelizacije. Projekt se uglavnom usredotočio na načelo podijeli-pa-vladaj za paraleliziranje algoritama sortiranja, ali postoje i druge tehnike kao što su map-reduce ili cjevovodni paralelizam koje bi se mogle istražiti.

REFERENCES

[1] Božidar D, Dobravec T. A comparison study between sequential and parallel sorting algorithms. https://github.com/darkobozidar/sequential-vs-parallel-sort, 2015.

[2] D. Zurek, M. Pietron, M. Wielgosz, and K. Wiatr, “Comparison of hybrid sorting algorithms implemented on different parallel hardware platforms,” Computer Science, vol. 14, no. 4, p. 679, 2013, issn: 2300-7036.

[3] K. Sujatha, P. V. N. Rao, A. A. Rao, V. G. Sastry, V. Praneeta, and R. K. Bharat, “Multicore parallel processing concepts for effective sorting and searching,” in 2015 International Conference on Signal Processing and Communication Engineering Systems, 2015, pp. 162–166. doi: 10.1109/SPACES.2015. 7058238.