Palyginkite aktoriu modelio ir monitoriu teikiamas priemones duomenu modifikavimui is keliu giju: ivardinkite kiekvieno is ju privalumas, kuriuo is ju programuojant lengviau suklysti ir kodel (3)

Monitoriu privalumas tai tas kad uzrakinus(sukurus krinite sekcija) beveik garantuojame kad gausime tokios duomenis kokiu norime. Suklysti galima labiau aktoriu modelyje. Monitoriais lengviau nes galima pazymeti kritine sekcija, kuri bus vykdoma su tarpusavio isskyrimu taipogi galima uzmigdyti gija/pazadinti uzmigdytas gijas lengvai.

Aktoriu modelyje pasto dezute paslepta nuo programuotojo, negalima nustatyti jos dydzio kaip sinchronizuotiems kanalams, reikia paciam uztikrinti kad nebus perpildoma pasto dezute ir jai uzteks atminties, jei pasto dezute tuscia o procesas bando apdoroti sekancia zinute, jis yra blokuojamas, kol ta zinute ateis, del sios priezasties gali susidaryti aklavietes situacija(del to mano nuomone monitoriais yra dirbti lengviau)

Palyginkite tarpusavyje java, openmp ir cuda gijas: ypatybes, panasumai skirtumai (3)

java, openmp ir cuda gijos tai java turi gijos objekta, kuris turi visus metodus gijos paleidimui ir pns, o cuda gijos objekto neturi, tiesiog isskiriama gpu atmintis resursams, rasoma device arba global prie funkcijos ir ta funkcija jau paleidziama gpu lygiagreciai, pati gpu giju valdyma galima atlikti paciu funkciju viduje, panasiai kaip ir openmp iskirtoje lygiagrecioje srityje (openmp taip pat nenaudojami giju objektai, vietoje to naudojama lygiagretumo iskyrimo direktyva, kuri paleidzia ir vykdo darba su gijomis), galima patikrinti/nurodyti paleistu giju kieki, paleistos gijos numeri

Tarkime, kad kompiuteris vykdo viena lygiagrecia programa, kiti procesai miega, kas vyksta kompiuteryje, kai vienu metu vykdome daugiau giju, nei kompiuteris turi branduoliu? Kokias atvejais gali buti prasminga paleisti daugiau giju, nei kompiuteris turi branduoliu? (4)

Jei aktyviu procesu/giju sistemoje veikia daugiau, nei yra cpu branduolius, vis tiek bus vykdomas uzduociu perjungimas, nors vienas procesas giju vykdo maziau, nei cpu branduoliu skaicius.

Procesu perteklius laukia operacines sistemos „darbu pasiskirstyme“ (task manager matoma) savo eiles.

Threadai laukia operacines sistemos kada operacine sistema leis jiems „judeti“

Prasminga paleisti daugiau giju nei kompiuteris turi branduoliu kai yra galimybe pvz., kad jeigu paleidi daugiau threadu, ir jei cpu nera apkrautas 100 procentu, todėl daugiau threadu paleistu „scheduled“ yra gera situacija mano manymu, nes cpu nera pilnai apkrautas.

Palyginkite aktoriu modelyje taikoma zinuciu apdorojima su ivykiu ciklais, naudojamais asinchroniniame programavime. Kuo jie panasus ir kuo skiriasi? (2)

Pašto dėžutės elgiasi kaip buferis, siuntėjas įrašo ir toliau tęsia darbą, o gavėjas nusiskaito kada reikia. Asinchroniniame programavime naudojamos funkcijos, kurios veikia kartu su main gija ir nestabdo main gijos darbo. Taipogi visa informacija laikoma Promise objekte. Manau lengviau kuri asinchronines funkcijas, nes galima naudoti async() ir await().

Palyginkite OpenMp ir Cuda giju paleidima, sinchronizacijos priemones. Kokie skirtumai, bendros atminties naudojimo taikant sias priemones? (4)

OpenMp gijos naudoja bendrąją atimintį (gali būti ir ne bendra jei naudojama private direktyva), o cuda turi kelis tipus: vietinė (matoma tik vienai gijai), bendroji (matoma visam blokui) ir globali (matoma visiems blokams) atmintys. Taipogi OpenMp naudoja RAM, o CUDA VRAM(GPU atmintį). Todėl atliekant darbą su CUDA pradinius duomenis iš RAM reikia nukopijuoti į VRAM su memcpy ir baigus darbą ir sulaukus cudaDeviceSynchronize() reikia nukopijuoti rezultatus i RAM iš VRAM. Gijų paleidimas panašus, skiriasi tik tai, kad OPENMP atveju reikia nurodyti legiagretumo sritį ir ten bus paleidžiamos gios. CUDA atveju reikia sukurti global funkciją, kurią CPU iškvies bet vykdys GPU. **Dar sinhronizacija reikia aprasyt**

Kas yra gijų telkinys (thread pool)? Kokie jo panaudojimo privalumai ir trūkumai?(5)

Gijų telkinys- panasus dalykas i openmp parallel for, galima nurodyti veiksmus, procesu skaiciu ir automatiskai viskas bus paskirstyta, skirtumas tik tas, kad openmp vietoj procesu naudojamos gijos.

Privalumai:

Panaudoja jau prieš tai sukurtas gijas.

Galima paskirti darbą, kurį gijos pasidalina pačios

Trūkumai:

Jei duomenys isdalinti nelygiai- vienos gijos skaiciuos ilgiau, kitos- trumpiau: skirtingų elementų, apdorojimo laikas gali būti skirtingas.

Kai gijų, per mažai, skaičiavimai vyksta lėčiau, kai gijų, per daug- vyksta konteksto perjungimas, gijų valdymas sunaudoja didele dalį skaičiavimo laiko.

Kokiais atvejais gali susidaryti aklavietė naudojant paskirstytos atminties modelį? Kaip to išvengti? (4)

Aklavietė susidarys kai bus siunčiama žinutė, kurios niekas nelaukia, arba bandoma priimti žinutę, kurios niekas nesiunčia. Kad to išvengti reikia įsitikinti, kad siunčiamų ir gaunamų žinučių kiekis vienodas (Go kanalų kurie priima ir kurie siunčia kiekis vienodas).

Jums duotas uždavinys pagreitinti programą. Lėčiausia programos dalis yra milžiniško masyvo sumavimas. Kokias lygiagretaus programavimo priemones naudosite ir kodėl? Kokią įtaką pasirinktas sprendimas turės programos palaikomumui, t. y., lyginant su nuosekliu variantu, kiek programos kodas taps sudėtingesnis skaityti? (2)

Naudosiu C++ OpenMP parallel for, ciklo viduj sudėsiu visas reikšmes ir gražinsiu rezultatų masyvą. OpenMP parallel for pats sinchronizuoja lygiagretinimą, todėl programa smarkiai nepasunkės, skaitomumas smarkiai nepasikeis.

Kokie duomenų apsikeitimo tarp procesų būdai galimi naudojant Python multiprocessing modulį? Palyginkite juos su Go kanalais, kokie panašumai ir skirtumai? (3)

Queue ir Pipe. Python multiprocessing.Queue sukurtas objektas gan panašus į Go buferizuotą kanalą (ten irgi susidaro kaip ir eilė ir iš eilės yra imami duomenys) skirtumas kad tam nenaudojami metodai (kaip python get ir put) bet naudojamos direktyvos <- /->, multiprocessing. Pipe skiriasi šiek tiek daugiau, ten sukuriami kanalo galai, kurie naudojami su send ir recv, kas yra šiek tiek panašiau į MPI, kas panašu į Go, tai, kad bandant į tą patį kanalą rašyti ar iš jo skaityti iš kelių procesų gali būti sugadinti duomenys (Go nebuferizuotam kanale).