DASTURNING SIKLIK BO'LIMLARINI OPTIMALLASHTIRISH

Sh. I. Xodiyev, O. B. Farmonov
O’zbekiston Milliy universiteti

ANNOTATSIYA

Transformatsiyalar orqali optimallashtirish algoritmlari va ularni amalga oshirish usullari sohasidagi ma'lum ishlanmalar ko'rib chiqildi. Sikl bo’limlarini transformatsiya yo’li bilan optimallashtirish uchun bazi algoritmlardan foydalanib Python dasturlash tilida skilni optimallashtirish dasturi tuzildi. Skilni optimallashtirishda Python dasturlash tilining kerakli kutubxonalaridan foydalanildi.

Kalit so’zlar: Transformatsiya, optimallashtirish, Python, protsessor arxitekturasi, kod/shakl.

KIRISH

Zamonaviy kompyuterlarning ortib borayotgan quvvati nafaqat dasturning tezroq bajarilishiga, balki yangi imkoniyatlarga ham olib keladi. Bundan tashqari, protsessor arxitekturasidagi o'zgarishlar avtomatik optimallashtirishning yangi usullarini yaratish va mavjud usullarni qayta ishlashni talab qiladi

Optimallashtirishni transformatsiya metodlari bilan amalga oshirsa bo’ladi. Transformatsiyalar - bu dasturning funksionalligini(biror ish bajara olishligini) o'zgartirmasdan uning yo’lini(ishlash usulini) o'zgartiradigan operatsiyalar(amallar). Ular dasturlarning ishlashini yaxshilash uchun dasturiy ta'minotni optimallashtirishda keng qo'llaniladi. Transformatsiyalar manba kodi (source code-dasturchilar tomonidan yozilgan ya’ni inson o’qiy oladigan kod), oraliq shakllar(intermediate presentations) (mashina o’qiy oladigan kod/shakl) va mashina kodi (ikkilik kod,komputerning protsessori orqali amalga oshiriladigan) kabi turli xil dastur shakllarida qo'llanilishi mumkin. Transformatsiya yo’li bilan optimallashtirishdan maqsad dasturlarning to'g'riligini saqlab, ularning ishlashini yaxshilashdir.

Transformatsiya yo’li bilan optimallashtirish computer science (kompyuter fanlari) va software engineering (dasturiy ta'minot muhandisligi) ning muhim bosqichiga aylandi. Ushbu maqolada transformatsiyalarni optimallashtirish algoritmlari ko’rib chiqildi. Asosan sikl optimallashtirishga doir turli xil transformatsiyalarni va ularning qo'llanilishi tasvirlab berildi.

Maqoladi sikl bo’limlarini transformatsiya yo’li bilan optimallashtirish uchun bazi algoritmlardan foydalanib Python dasturlash tilida skilni optimallashtirish ko’rib chiqildi. Skilni

optimallashtirishda Python dasturlash tilining **AST** (**Abstract Syntax Tree**), **ASTOR**, **NumPy**, **Numba**, **Cython**, **Intel MKL** kutubxonalaridan foydalanildi.

ADABIYOTLAR TAHLILI VA METODOLOGIYA

Python-da AST (Abstract Syntax Tree) va ASTOR ikkita bog'liq kutubxona bo'lib, ular Python manba kodini boshqarish va o'zgartirish uchun ishlatiladi. **AST** o'rnatilgan Python moduli bo'lib, u Python dasturining strukturasini tugunlar daraxti sifatida ko'rsatish yo'lini ta'minlaydi, bu erda har bir tugun funktsiya chaqiruvi, o'zgaruvchilarni tayinlash yoki sikl kabi dasturning bir qismini ifodalaydi. Ushbu daraxtni dasturiy jihatdan tahlil qilish yoki o'zgartirish osonroq bo'lishi mumkin bo'lgan manba kodining soddalashtirilgan versiyasi sifatida qarash mumkin.

ASTOR tashqi Python kutubxonasi bo‘lib, AST moduliga asoslanadi va ASTdan Python kodini yaratish, mavjud ASTni o‘zgartirish yoki ASTni turli yo‘llar bilan tahlil qilish uchun yordamchi dasturlar to‘plamini taqdim etadi. ASTOR avtomatik tarzda qayta ishlash, optimallashtirish yoki yangi kodni yaratish kabi kod o'zgarishlarini amalga oshirish uchun ishlatilishi mumkin.

AST ham, **ASTOR** ham statik tahlil, kod ishlab chiqarish va refaktoringni amalga oshiradigan vositalarda keng qo'llaniladi. Ular, ayniqsa, IDE, linters, kod formatlovchilari va kod generatorlari kabi manba kodini manipulyatsiya qilish talab qilinadigan ishlab chiqish muhitlarida foydalidir.

Siklni optimallashtiruvchi algoritmlar yani **Loop Unrolling** , **Loop Fusion**, **Loop Distribution**, **Loop Alignment**, **Loop Interchange** algoritmlar qo’llanildi va sikl qatnashgan kodni Python dasturlash tili yordamida optimallashgan ko’rinishga keldirildi. Bundan tashqari yuqoridagi algoritmlarni qanday ishlatish haqida ham malumotlar va dasturlar keltirildi. **AST** va **ASTOR** kutubxonalari Python kodining mavhum sintaksis daraxti (AST) bilan ishlash orqali sikllarni optimallashtirish uchun mustahkam asos yaratadi. Ular sizga kodni **AST**ga tahlil qilish, AST tugunlarini kesib o'tish va o'zgartirish va o'zgartirilgan **AST** dan kod yaratish imkonini beradi. Ko'pgina hollarda, **AST** va **ASTOR** sikl optimallashtirishni amalga oshirish uchun etarli. Biroq, siz qo'llamoqchi bo'lgan optimallashtirish usullarining murakkabligiga yoki kodingizning o'ziga xos talablariga qarab, siz qo'shimcha kutubxonalar yoki vositalardan foydalanishingiz mumkin. Quyida shunday kutubxonalar keltirilgan.

NumPy. Agar sizning kodingiz raqamli hisoblashlar va massiv operatsiyalarini o'z ichiga olsa, NumPy-dan foydalanish unumdorlikni sezilarli darajada oshirishga olib kelishi mumkin. NumPy samarali massiv operatsiyalari, translyatsiya va vektorlashtirilgan hisoblashlarni ta'minlaydi, bu ko'pincha aniq sikllarni almashtirishi mumkin.

Numba. Numba raqamli hisoblashlarni optimallashtirishga ixtisoslashgan Python uchun o'z vaqtida(just-in-time) (JIT) kompilyatoridir. Bu **'@jit'** bilan funksiyalarni avtomatik ravishda yuqori samarali mashina kodiga kompilyatsiya qilish imkonini beradi. Numba sikllarni optimallashtirishi va raqamli hisoblarni tezlashtirishi mumkin.

Cython. Cython - bu Pythonni C ga o'xshash statik yozish bilan kengaytiradigan dasturlash tili. Bu sizga Pythonga o'xshash sintaksisda kod yozish imkonini beradi va uni C yoki C++ kengaytmalariga kompilyatsiya qilish imkoniyatini beradi. Cython yuqori darajada optimallashtirilgan mahalliy kodni yaratish orqali sikllarni optimallashtirishi va ish faoliyatini yaxshilashi mumkin.

Intel MKL. Intel Math Kernel Library (MKL) ilmiy hisoblash uchun yuqori darajada optimallashtirilgan matematik funktsiyalar to'plamidir. U turli xil matematik operatsiyalarni, jumladan, matritsa operatsiyalarini, Tez Furye Transformatsiyalarini va boshqalarni optimallashtirilgan amalga oshirishni ta'minlaydi. MKL dan foydalanish ma'lum raqamli hisob-kitoblarda unumdorlikni oshirishi mumkin.

Bular sikl optimallashtirish uchun AST va ASTORni to'ldirishi mumkin bo'lgan kutubxonalar va vositalarning bir nechta misollari. Qo'shimcha kutubxonalar yoki vositalarni tanlash kodingizning o'ziga xos talablariga va qo'llamoqchi bo'lgan optimallashtirish usullariga bog'liq. Kodingizni tahlil qilish, ishlashdagi qiyinchiliklarni aniqlash va shunga mos ravishda tegishli vositalarni tanlash muhimdir. [7]

Transformatsiya turlari. Dasturiy ta'minotni optimallashtirish uchun ishlatilishi mumkin bo'lgan har xil turdagi transformatsiyalar mavjud. Eng ko'p ishlatiladigan transformatsiyalardan ba'zilari:

Loop Transformations (Sikl transformatsiyasi**).** Loop(Sikl) transformatsiyalari dasturlarda sikllarni optimallashtirish uchun ishlatiladi.Loop transformatsiyalari uchun ko'plab algoritmlar mavjud.

1. **Loop(Sikl)ni ochish** (loop unrolling). Kerakli takrorlashlar sonini kamaytirish uchun halqa tanasini takrorlaydi va shu bilan halqani boshqarish yukini kamaytiradi.

Loop optimization uchun 1-metod **loop unrolling**.

```
def loop_unrolling_optimization(loop_body, unroll_factor):
unrolled_loop_body = [ ]
for i in range(0, len(loop_body), unroll_factor):
unrolled_loop_body.extend(loop_body[i:i+unroll_factor])
return unrolled_loop_body
```

Kod ikkita parametrni qabul qiluvchi “loop_unrolling_optimization” funksiyasini belgilaydi: “loop_body” va “unroll_factor”. Ushbu funktsiyaning maqsadi ma'lum bir sikl tanasida siklni ochishni optimallashtirishni amalga oshirishdir.

Funktsiya ichida “unrolled_loop_body” deb nomlangan bo'sh ro'yxat yaratiladi. Keyin, “unroll_factor” qadam o'lchami bilan “loop_body” indekslarini takrorlash uchun for sikli ishlatiladi. Bu shuni anglatadiki, sikl bir vaqtning o'zida “unroll_factor” elementlarini qayta ishlaydi. Loopning har bir iteratsiyasida kod “loop_body” bo'lagini “unrolled_loop_body” ro'yxatiga qo'shadi. Bo'lim joriy indeksdan i dan “i + unroll_factor” ga olinadi, “unroll_factor” elementlarini “loop_body” dan samarali ajratib oladi.

Nihoyat, funksiya optimallashtirish qo'llanilgandan so'ng sikl tanasini o'z ichiga olgan “unrolled_loop_body” ro'yxatini qaytaradi.

Loop transformatsiyalari qo'shimcha xarajatlarni kamaytirish va xotiraga kirish usullarini yaxshilash orqali dasturlarning ishlashini yaxshilashi mumkin. [1]

2. **Loop Fusion** (Skilni birlashtirish). Keshdan(Komputer xotirasida saqlanib qolgan narsalar) foydalanishni yaxshilash va qo'shimcha xarajatlarni kamaytirish uchun o'xshash operatsiyalarni bajaradigan bir nechta sikllarni bitta siklga birlashtiradi. Kod ikkita parametrni qabul qiluvchi “loop_fusion_optimization” funksiyasini belgilaydi: “loop1_body” va “loop2_body”. Ushbu funktsiyaning maqsadi ikkita berilgan sikl tanasida sikl sintezini optimallashtirishni amalga oshirishdir.Funktsiya ichida + operatori yordamida “loop1_body” va “loop2_body” ni birlashtirish orqali “fused_loop_body” deb nomlangan yangi ro'yxat yaratiladi. Bu operatsiya ikkala halqa tanasining elementlarini bitta ro'yxatda birlashtiradi.

Va nihoyat, funktsiya optimallashtirish qo'llanilgandan so'ng birlashtirilgan sikl tanasini ifodalovchi “fused_loop_body” ni qaytaradi. [6]

3. **Loop distribution**.

```
def loop_distribution_optimization(loop_body, distribution_factor):
distributed_loop_body = []
for statement in loop_body:
distributed_loop_body.extend([statement] * distribution_factor)
return distributed_loop_body
```

Bu funksiya sikl statementlar ro‘yxatini (loop_body) va distribution_faktor ni oladi. Loop_body har bir bayonotni ko'rsatilgan distribution_faktor ga ko'ra takrorlash orqali yangi ro'yxatni (distributed_loop_body) yaratadi. Natijada paydo bo'lgan ro'yxatda distribution_faktor ga asoslangan takroriy ko'paytirilgan sikl statementlari mavjud. [2]

4. **Loop alignment.**

```
def loop_alignment_optimization(loop_body, alignment_factor):
alignment = len(loop_body) % alignment_factor # alignment factorni hisoblash
if alignment != 0:
alignment_body = loop_body[-alignment:] # alignment tanasini ajratib olish
loop_body.extend(alignment_body) # loop tanasini alignment tanasi bilan kengaytirish
return loop_body
```

Ushbu kodda ‘loop_alignment_optimization’ funksiyasi kirish sifatida ‘loop_body’ va ‘alignment_factorni’ oladi. U sikl tanasining uzunligini tekislash koeffitsientiga bo'lishning qolgan qismini topib, tekislash koeffitsientini hisoblab chiqadi. Agar tekislash nolga teng bo'lmasa, u sikl tanasining oxiridan tekislash tanasini chiqaradi va tekislash tanasini qo'shish orqali sikl tanasini kengaytiradi. Nihoyat, u tekislangan sikl tanasini qaytaradi.

E'tibor bering, alignment factor kerakli tekislash o'lchamini yoki chegarasini ifodalaydi va u muayyan optimallashtirish texnikasi yoki dastur talablariga bog'liq bo'lishi mumkin. [5]

5. **Loop Interchange** (Siklni o’zaro almashinuvi).

Codening umumiy ko’rinishi

```
def loop_interchange_optimization(loop1_body, loop2_body):
new_loop_body = [ ]
for i in range(len(loop2_body)):
for j in range(len(loop1_body)):
new_loop_body.append(loop2_body[i] + loop1_body[j])
return new_loop_body
```

Ma'lumotlar joylashuvini yaxshilash va kesh o'tkazib yuborilishini kamaytirish uchun ichma ich joylashgan Ushbu kod “loop_interchange_optimization” funksiyasi ikkita parametrni oladi: loop1_body va loop2_body.Funktsiya ichida, sikl almashinuvi natijasida paydo bo'lgan yangi sikl tanasini saqlash uchun “new_loop_body” deb nomlangan bo'sh ro'yxat yaratiladi. “loop2_body” va “loop1_body” bo'yicha takrorlash uchun ikkita ichki o'rnatilgan for sikl ishlatiladi. "loop2_body” va “loop1_body” elementlarining har bir kombinatsiyasi uchun kod “append” funktsiyasidan foydalanib, ularning yig'indisini (birlashtirish) “new_loop_body” ro'yxatiga qo'shadi. Nihoyat, funktsiya sikl almashinuvini optimallashtirish qo'llanilgandan so'ng, sikl tanasini ifodalovchi “new_loop_body” ni qaytaradi. [4]

6. **Loop-Invariant Code Motion** . Siklda bajariladigan ko'rsatmalar sonini kamaytirish uchun sikl indeksiga bog'liq bo'lmagan kodni sikldan tashqariga ko'chiradi.

```
def loop_invariant_code_motion_optimization(loop_body):
invariant_code = [ ]
new_loop_body = [ ]
for stmt in loop_body:
if is_loop_invariant(stmt):
invariant_code.append(stmt)
else:
new_loop_body.extend(invariant_code)
invariant_code = [ ]
new_loop_body.append(stmt)
new_loop_body.extend(invariant_code)
return new_loop_body
def is_loop_invariant(stmt):
return True or False
```

Ushbu kodda bizda ikkita funktsiya mavjud. loop_invariant_code_motion_optimization funksiyasi kirish sifatida loop_body ni oladi, bu original sikl tanasini ifodalaydi.

Funktsiya ichida ikkita ro'yxat yaratiladi: “loop-invariant” kodini saqlash uchun “invariant_code” va “loop-invariant” kod harakatini optimallashtirishdan keyin yangi sikl tanasini saqlash uchun “new_loop_body”, “loop_body” dagi har bir holat takrorlanadi. Agar statement “loop_invariant” funksiyasi asosida “loop-invariant” ekanligi aniqlansa, u “invariant_code” ro'yxatiga qo'shiladi. Aks holda , ” invariant_code” da saqlangan “loop-invariant” kodi, uni “new_loop_body” ga qo'shish orqali sikldan tashqariga o'tkaziladi. Keyin “invariant_kodlar” ro'yxati qayta o'rnatiladi.

Loopdan keyin “invariant_code” ro'yxatidagi qolgan har qanday siklning o'zgarmas kodi uni “new_loop_body” ga qo'shish orqali sikldan tashqariga ko'chiriladi.

Va nihoyat, funktsiya “loop-invariant kod” harakatini optimallashtirish qo'llanilgandan so'ng, sikl tanasini ifodalovchi “new_loop_body” ni qaytaradi.

“is_loop_invariant” funksiyasi berilgan statementning “loop-invariant” yoki yo'qligini aniqlaydigan to'ldiruvchidir. Agar ibora “loop-invariant” bo'lsa, u "True", aks holda "False" ni qaytarishi kerak. [8]

7.**Strength reduction** (Quvvatni pasaytirish) . Qimmatbaho operatsiyalarni (ko'paytirish yoki bo'lish kabi) arzonroq ekvivalent operatsiyalar bilan almashtiradi (masalan, qo'shish yoki bitni almashtirish).

Bu yerda Strength Reduction(Quvvatni pasaytirish) metodi uchun umumiy algorithm(kod)

```
def strength_reduction_optimization(loop_body):
new_loop_body = []
for stmt in loop_body:
if is_expensive_op(stmt):
new_stmt = replace_expensive_op_with_cheap_op(stmt)
new_loop_body.append(new_stmt)
else:
new_loop_body.append(stmt)
return new_loop_body
def is_expensive_op(stmt):
return True or False
def replace_expensive_op_with_cheap_op(stmt):
return new_stmt
```

Ushbu kodda bizda uchta funktsiya mavjud. ”strength_reduction_optimization” funksiyasi kirish sifatida “loop_body” ni oladi. Funktsiyaning ichida quvvatni pasaytirishni optimallashtirishdan so'ng “new loop body” ini saqlash uchun “new_loop_body” deb nomlangan bo'sh ro'yxat yaratiladi. Statement “loop_body” dagi har bir statement ustida takrorlanadi.

Har bir statement uchun kod “is_expensive_op” funksiyasidan foydalangan holda statementni qimmat operatsiya mavjudligini tekshiradi. Agar u qimmat operatsiya ekanligi aniqlansa , kod qimmat operatsiyani arzonroq operatsiya bilan almashtiradi. Natijada yangi statement “new_loop_body” ga qo'shiladi.

Agar statement qimmat operatsiyani o'z ichiga olmasa, original statement new_loop_body ga avvalgidek qo'shiladi. Va nihoyat, funktsiya quvvatni pasaytirishni optimallashtirish qo'llanilgandan so'ng sikl tanasini ifodalovchi new_loop_body ni qaytaradi.

“is_expensive_op” funksiyasi berilgan statementda qimmat operatsiya mavjudligini aniqlaydigan to'ldiruvchidir. Agar statement qimmat operatsiyani o'z ichiga olsa, u "True" ni, aks holda "False" ni qaytarishi kerak.

“replace_expensive_op_with_cheap_op” funksiyasi malum statementdagi qimmat operatsiyani arzonroq operatsiya bilan almashtiruvchi to'ldiruvchidir. O'zgartirishni amalga oshirish va

yangi statementni qaytarish uchun kerakli amalni amalga oshirishi kerak. [strength_reduction] [6]

8. **(Loop Vectorization)** Loop vektorizatsiyasi. Zamonaviy protsessorlarda mavjud bo'lgan parallelizmdan foydalanish uchun bir vaqtning o'zida siklning bir nechta iteratsiyasini qayta ishlash uchun vektor ko'rsatmalaridan foydalanadi.

Bu yerda Loop Vectorization metodi uchun umumiy metod(kod) ko’rsatilgan

```
import numpy as np
def vectorize_loop(loop_body, vector_size):
n = len(loop_body)
# Vektorlashtirishdan keyin qancha iteratsiya qolishini aniqlang
remainder = n % vector_size
# Vektorizatsiya bilan mos keladigan yangi sikl tanasini yaratish
new_loop_body = loop_body[: n - remainder]
# Vektorlashtirilgan halqa tanasini yarating
for i in range(0, n - remainder, vector_size):
# Loop tanasining bir bo'lagini ajratib oling
slice_body = loop_body[i: i + vector_size]
# Bo'lakni vektor operatsiyasiga aylantirish
vector_op = np.array(slice_body)
# Yangi sikl tanasiga vektor operatsiyasini qo'shing
new_loop_body.append(vector_op)
# Qolgan statementlarni yangi sikl tanasiga qo'shing
new_loop_body.extend(loop_body[n – remainder :])
return new_loop_body
```

Ushbu kodda “vektorize_loop” funksiyasi ikkita parametrni oladi: original sikl tanasini ifodalovchi “loop_body” va vektorlashtirilgan bo'laklarning o'lchamini belgilaydigan “vector_size”.

Funktsiya ichida “loop_body” uzunligi hisoblab chiqiladi va n o'zgaruvchisida saqlanadi. Qolgan o'zgaruvchi vektorizatsiyadan keyin qolgan iteratsiyalar sonini aniqlash uchun ishlatiladi.

“new_loop_body” deb nomlangan yangi ro'yxat original “loop_body”ni n - qoldig'igacha bo'lish orqali yaratiladi. Bu yangi sikl tanasi vektorizatsiya bilan mos kelishini ta'minlaydi.

Keyin sikl 0 dan n gacha - qoldiq oralig'ida “vektor_size” qadam o'lchami bilan takrorlanadi. Har bir iteratsiyada vektor operatsiyasi sifatida qayta ishlanishi kerak bo'lgan elementlarning bir qismini ifodalovchi “loop_body” bo'limi chiqariladi.

“create_vector_operation” funksiyasi (kodda ko'rsatilmagan) elementlarning bir qismini vektor operatsiyasiga aylantirish jarayonini ifodalovchi to'ldiruvchidir. Ushbu operatsiya vektorlashtirilgan ko'rsatmalar yoki vektorizatsiyaga xos kutubxonalardan foydalanishni o'z ichiga olishi mumkin.

Natijada “vektor_op” vektor operatsiyasi keyin “new_loop_body” ga qo'shiladi.

Loopdan keyin “loop_body” dan qolgan statementlar “new_loop_body” ga qo'shiladi. Nihoyat, funktsiya vektorlashtirishni optimallashtirish qo'llanilgandan so'ng tsikl tanasini ifodalovchi “new_loop_body” ni qaytaradi. [3]

9. **Loop Tiling.** Ma'lumotlar joylashuvini yaxshilash va kesh o'tkazib yuborilishini kamaytirish uchun katta halqani keshga mos keladigan kichikroq halqalarga ajratadi.

Bu yerda **Loop Tiling** uchun umumiy code:

Ushbu kodda “tile_loop” funksiyasi ikkita parametrni oladi: original halqa tanasini ifodalovchi “loop_body” va har bir tile hajmini belgilaydigan “tile_size”

Funktsiya ichida tilelar soni (num_tiles) “loop_body” uzunligini “tile_size” ga bo’lish yo'li bilan hisoblanadi. Bu hosil bo'lishi mumkin bo'lgan to'liq tilelar sonini aniqlaydi

Yangi sikl tanasini tile bilan saqlash uchun “tiled_loop_body” deb nomlangan yangi ro'yxat yaratiladi.

Tsikl 0 dan ‘num_tiles’ oralig'ida takrorlanadi. Har bir tile uchun boshlang'ich va tugatish indekslari ‘tile_size’ asosida aniqlanadi. Joriy tilega mos keladigan statementlar slicing yordamida “loop_body” dan chiqariladi.

Chiqarilgan tile statementlari har bir statementni alohida-alohida yangi sikl tanasiga qo'shadigan kengaytirish funksiyasi yordamida “tiled_loop_body” ga qo'shiladi.

Tsikldan so'ng, to'liq tile hosil qila olmagan qolgan barcha statementlar “tiled_loop_body” ga qo'shiladi.

Va nihoyat, funktsiya ‘tiled_loop_body’ qiymatini qaytaradi, bu esa tile qo'yish optimallashtirish qo'llanilgandan so'ng tsikl tanasini ifodalaydi.

```
def tile_loop(loop_body, tile_size):
num_tiles = len(loop_body) // tile_size
# tilelar bilan yangi sikl tanasini yaratish
tiled_loop_body = []
for tile_idx in range(num_tiles):
# Oldingi tile uchun boshlang’ich va oxirgi indexlarni aniqlash
start_idx = tile_idx * tile_size
end_idx = (tile_idx + 1) * tile_size
tile_statements = loop_body[start_idx:end_idx]
```

```
tiled_loop_body.extend(tile_statements)
tiled_loop_body.extend(loop_body[num_tiles*tile_size:])
return tiled_loop_body [loop_tile]
```

10.**Loop Strip-Mining**. Yuqori xarajatlarni kamaytirish va keshdan foydalanishni yaxshilash uchun halqani kichikroq bo'laklarga ajratadi.

Bu yerda Loop Strip-Mining uchun umumiy algoritm(kod) keltirilgan. Ushbu kodda “strip_mine_loop” funksiyasi ikkita parametrni oladi: original sikl tanasini ifodalovchi “loop_body” va har bir strip o'lchamini belgilaydigan “strip_size”.

Funktsiyaning ichida striplar soni (chiziqlar soni) sikl_tanasi uzunligini “strip_size”ga bo'lish yo'li bilan hisoblanadi. Bu hosil bo'lishi mumkin bo'lgan to'liq chiziqlar sonini aniqlaydi. “stripped_loop_body” deb nomlangan yangi ro'yxat strip mining bilan yangi sikl tanasini saqlash uchun yaratilgan.

Loop 0 dan “num_strips” oralig'ida takrorlanadi. Har bir strip uchun boshlang'ich va tugatish indekslari “strip_size” asosida aniqlanadi. Joriy strippga mos keladigan statementlar slicing yordamida “loop_body” dan chiqariladi.

“Strip_loop” deb nomlangan satr o'zgaruvchisi chiziqli halqa tanasini qurish uchun yaratilgan. U tarmoqli indeksi (j_strip_idx) uchun strip o'lchami bo'ylab takrorlanadigan for loop statementi bilan boshlanadi. Keyin stripdagi har bir statement ichkariga kiritiladi va satrlarni birlashtirish va birlashtirish operatsiyasidan foydalangan holda “strip_loop” qatoriga qo'shiladi.

Olingan chiziqli halqa tanasini ifodalovchi ‘strip_loop’ satri ‘stripped_loop_body’ ga qo'shiladi.

Sikldan so'ng, to'liq strip hosil qila olmagan qolgan barcha statementlar “stripped_loop_body” ga qo'shiladi.

Nihoyat, funktsiya strip mining optimallashtirish qo'llanilgandan keyin halqa tanasini ifodalovchi “stripped_loop_body” ni qaytaradi.

```
def strip_mine_loop(loop_body, strip_size):
# Kerakli bo’laklar sonini aniqlash
num_strips = len(loop_body) // strip_size
# Strip bilan yangi sikl tanasini yaratish
stripped_loop_body = [ ]
for strip_idx in range(num_strips):
# Avvalgi strip uchun bosh va oxiri idexni aniqlash
start_idx = strip_idx * strip_size
end_idx = (strip_idx + 1) * strip_size
strip_statements = loop_body[start_idx:end_idx]
strip_loop = f"for j_{strip_idx} in range({strip_size}):\n"
```

```
strip_loop += '\n'.join([f" {s}" for s in strip_statements])
# Stripni yangi tsiklga qo’sh
stripped_loop_body.append(strip_loop)
stripped_loop_body.extend(loop_body[num_strips*strip_size:])
return stripped_loop_body [6]
```

11. **Loop Reversal.** Keshdan foydalanishni yaxshilash uchun aylanish tartibini o'zgartiradi.

Loopni teskari aylantirish siklni optimallashtirish texnikasi bo'lib, siklni takrorlash tartibini o'zgartiradi.

Bu yerda Loop Reversal uchun umumiy algorithm(kod)yozilgan:

```
def loop_reversal_optimization(loop_body):
return loop_body[: : -1]
```

Ushbu kodda “loop_reversal_optimization” funksiyasi kirish sifatida “loop_body” ni oladi.

[: : -1] belgisi “loop_body” ro'yxatidagi elementlar tartibini o'zgartirish uchun ishlatiladi. Start : stop : step ni [: : -1] sifatida belgilash orqali funktsiya teskari tartibda “loop_body” elementlarini o'z ichiga olgan yangi ro'yxatni qaytaradi.

Bu funksiya kirish sifatida sikl tanasini oladi va uning takrorlanishi teskari bo'lgan bir xil sikl tanasini qaytaradi

Misol uchun, agar loop_body [1, 2, 3, 4] bo'lsa, funktsiya [4, 3, 2, 1] ni qaytaradi, bu siklni teskari optimallashtirish qo'llanilgandan so'ng sikl tanasini ifodalaydi.

Statementlar tartibini teskari o'zgartirish orqali, siklni teskari optimallashtirish ba'zan ishlashni yaxshilashi yoki teskari tartib bilan yaxshiroq ishlaydigan maxsus optimallashtirishni yoqishi mumkin. [8]

12. **Parallelization**(Parallelizatsiya): Ko'p yadroli protsessorlarda mavjud parallellikdan foydalanish uchun tsiklni bir nechta bo’g’in yoki jarayonlarga ajratadi.

Parallelizatsiya - siklni optimallashtirish usuli bo'lib, u bir nechta protsessorlar yoki yadrolarda bir vaqtning o'zida bir nechta protsessorlarda yoki yadrolarda siklni takrorlashni amalga oshirishga imkon beradi va bajarish vaqtini qisqartirish orqali ishlashni yaxshilaydi. Pythonning multiprocessing moduli yordamida siklni parallellashtirish uchun umumiy kod:

Ushbu kodda “parallel_loop_optimization” funksiyasi ikkita parametrni oladi: oroginal sikl tanasini ifodalovchi ‘loop_body’ va parallellashtirish uchun ishlatiladigan jarayonlar sonini ko'rsatuvchi ‘num_procs’.

“chunk_size” ‘loop_body’ uzunligini ‘num_procs’ ga bo'lish yo'li bilan hisoblanadi. Bu har bir jarayon bajariladigan sikl iboralari sonini aniqlaydi.
Har bir parallel jarayon uchun Process obyektlarini saqlash uchun ‘procs’ deb nomlangan ro'yxat yaratiladi.

Keyin, ‘num_procs’ jarayonlarini yaratish uchun sikl ishlatiladi. Har bir jarayon uchun boshlang'ich va tugatish indekslari ‘chunk_size’ asosida aniqlanadi. Har bir jarayonning maqsadi argument sifatida ‘loop_body’, start va end ni qabul qiluvchi ‘execute_loop_chunk’ funksiyasiga o'rnatiladi.
‘execute_loop_chunk’ funksiyasi ichida sikl ko'rsatilgan diapazondagi (boshidan oxirigacha - 1) sikl statementlarini takrorlash uchun ishlatiladi. Bu jarayon bajariladigan sikl statementlarining bir qismini ifodalaydi. Sharhni parallel ravishda bajarilishi kerak bo'lgan haqiqiy sikl logic i bilan almashtirishingiz mumkin.
Jarayonlar proc.start() yordamida boshlanadi va Process ob'ektlari procs ro'yxatiga qo'shiladi.

Barcha jarayonlarni yaratgandan so'ng, har bir jarayon uchun proc.join() ni chaqirish uchun boshqa sikl ishlatiladi. Bu davom etishdan oldin asosiy jarayon barcha child processlarning tugashini kutishini ta'minlaydi.
Loop bayonotlarini bir nechta jarayonlarga taqsimlash va ularni parallel ravishda bajarish orqali parallel siklni optimallashtirish siklning umumiy bajarilish vaqtini yaxshilashga qaratilgan.

```
import multiprocessing
def parallel_loop_optimization(loop_body, num_procs):
chunk_size = len(loop_body) // num_procs
procs = []
for i in range(num_procs):
start = i * chunk_size
end = (i+1) * chunk_size if i != num_procs-1 else len(loop_body)
proc = multiprocessing.Process(target=loop_body[start:end])
proc.start()
procs.append(proc)
for proc in procs:
proc.join() [3]
```

REFERENCES

1. Meisam Booshehri, Abbas Malekpour, Peter Luksch . An Improving Method for Loop Unrolling . International Journal of Computer Science and Information Security , Vol. 11 , No.5 , May 2013 .

2. Alfred V. Aho, Ravi. Sethi, Jeffrey D. Ullman .Compilers, principles, techniques, and tools . 1986. ISBN 0-321-48681-1. 1035 pages.
3. Арипов М.М., Ходиев Ш.И. Методы трансляции и преобразования программ. Учебное пособие для ВУЗов. Ташкент, Университет, 2008. - 134 с.
4. https://arxiv.org/ftp/arxiv/papers/1308/1308.0698.pdf
5. https://ieeexplore.ieee.org/document/8117057
6. https://www.codingninjas.com/codestudio/library/loop-optimization
7. https://learnpython.com/blog/popular-python-libraries/
8. https://www.geeksforgeeks.org/loop-optimization-in-compiler-design/