

# Параллел алгоритмын зохиомж

F.CS306 ПАРАЛЛЕЛ ПРОГРАММЧЛАЛ – Лекц 4

Г.ГАНБАТ  
[ganbatg@must.edu.mn](mailto:ganbatg@must.edu.mn)

# Хичээлийн агуулга

- ❖ Amdahl болон Gustafson-ны хуулиудыг параллел програмын боломжилт хурдсалтын дээд хязгаарыг тодорхойлоход хэрэглэх
- ❖ Мөн эдгээр хуулиудын илүү өргөтгөсөн хурдсгалтын хэлбэртэй онцгой тохиолдлыг олох
- ❖ Хуваарилагдсан санах ойн паралелизмыг боломжийг судлах зорилгоор параллел програм зохиоход Foster-ын аргачлалыг хэрэгжүүлэх

# Amdahl-ын хууль

- ❖ speedup хэмжээг тооцоолох томъёог Amdahl's Law гэдэг
- ❖ Параллел ажиллагаанд хичнээн процессор ашиглагдаж байгаас үл хамааран программын speedup дараалсан кодын хэсгээс болж хязгаарлагадана.
- ❖ Бараг бүх програмд параллизмд өртөхгүй кодын хэсэг байдаг. Энэ нь параллель ажиллаж байсан ч гэсэн зөвхөн нэг процессороор ажиллуулах шаардлагатай кодын хэсэг юм.
- ❖ Хүрч болох speedup-ын дээд хязгаар (scaled speedup) өгнө.

# Amdahl-ын хуулийн гаргалгаа

$$T(1) = T_{\text{ser}} + T_{\text{par}}$$

програмын параллелизмаас ашиг  
хүртэж чадахгүй байгаа хэсэг

програмын параллелизмаас  
ashiг хүртэж байгаа хэсэг

- ❖ Эхлээд бид хамгийн боломжит speedup-ыг шугаман байна гэж үзнэ
- ❖ Дараа нь бид хүрч болох speedup-ын дээд хязгаарыг тооцоолно

$$T(p) \geq T_{\text{ser}} + \frac{T_{\text{par}}}{p}$$

$$S(p) = \frac{T(1)}{T(p)} \leq \frac{T_{\text{ser}} + T_{\text{par}}}{T_{\text{ser}} + \frac{T_{\text{par}}}{p}}$$

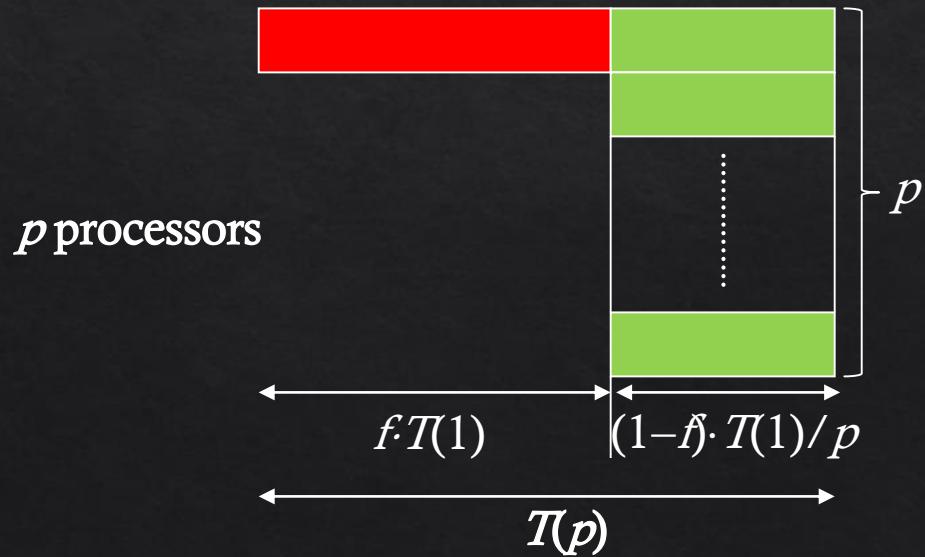
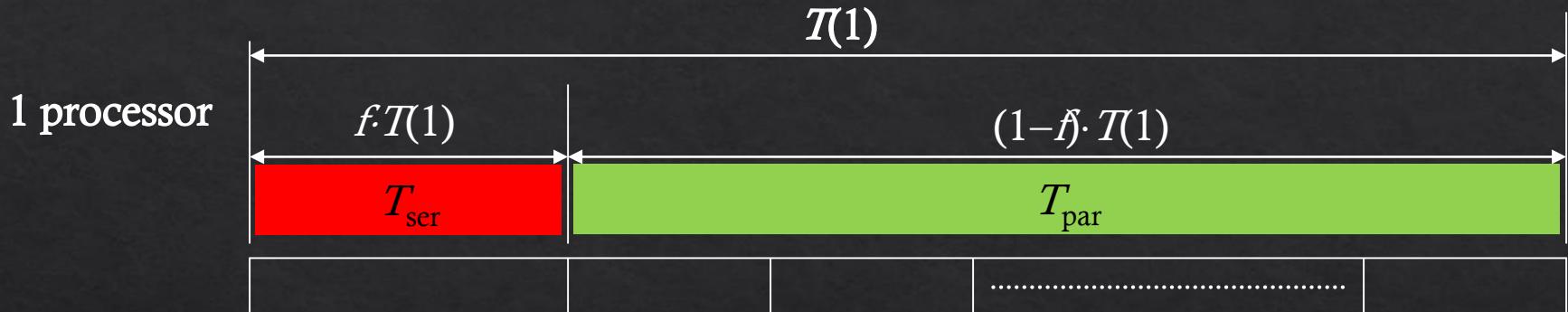
# Amdahl-ын хуулийн гаргалгаа

$$T_{\text{ser}} = f \cdot T(1) \text{ and } T_{\text{par}} = (1 - f) \cdot T(1); \quad (0 \leq f \leq 1)$$

- ❖  $T_{\text{ser}}$  ба  $T_{\text{par}}$  абсолют хэмжигдэхүүний оронд харгалзах хэсгийн fraction  $f$ -г хэрэглэе
- ❖ Дээд хязгаарийг олох өмнх томьёондоо орлуулвал Amdahl-ын хууль гарна.; Жны. speedup-ын дээд хязгаар нь зөвхөн  $f$  ба  $p$ -ээс хамаарна.

$$S(p) = \frac{T(1)}{T(p)} \leq \frac{T_{\text{ser}} + T_{\text{par}}}{T_{\text{ser}} + \frac{T_{\text{par}}}{p}} = \frac{f \cdot T(1) + (1 - f) \cdot T(1)}{f \cdot T(1) + \frac{(1 - f) \cdot T(1)}{p}} = \frac{1}{f + \frac{(1 - f)}{p}}$$

# Amdahl-ын хууль



$$S(p) = \frac{T(1)}{T(p)} \leq \frac{f \cdot T(1) + (1-f) \cdot T(1)}{f \cdot T(1) + \frac{(1-f) \cdot T(1)}{p}}$$

$$= \frac{f + (1-f)}{f + (1-f)/p} = \frac{1}{f + (1-f)/p}$$

# Amdahl-ын хууль

$$S(p) \leq \frac{1}{f + \frac{(1-f)}{p}}$$

- ❖  $f$ -ээр нэг процессор дээр дараалан гүйцэтгэгдэх үйлдлүүдийн хэсгийг тэмдэглэвэл  $(1-f)$  нь параллел ажиллах үйлдлүүдийн хэсгийг илэрхийлнэ.
- ❖ Кодын дараалсан хэсэг  $f$  нь бүхэл биш 0-ээс 1-ийн хооронд байна.
- ❖ Параллел программуудын чадавхийг урьдчилан тодорхойлохын тул Amdahl-ын хуулийн ашиглгаж болно

# Amdahl'-ын хууль – Жишээ

1. Програмын гүйцэтгэлийн 95% нь давталт дотор хийгддэг. 6 процессор дээр хэрэгжүүлсэн програмын параллел хувилбарын хамгийн их speedup хэд байх вэ?

$$S(6) \leq \frac{1}{0.05 + \frac{(1 - 0.05)}{6}} = 4.8$$

2. Програмын гүйцэтгэлийн хугацаны 10% нь дараалсан кодод зарцуулагддаг. Програмыг параллел ажиллулахад speedup-ийн хүрэх хязгаар хэд вэ?

$$S(\infty) \leq \lim_{p \rightarrow \infty} \frac{1}{0.1 + \frac{(0.9)}{p}} = 10$$

# Scaled Speedup

- ❖ **Amdahl-ын хуулийн хязгаарлалт:** асуудлын хэмжээ тогтмол бөгөөд ялгаатай прорцессорын тоо байх үед биелнэ ( $\Rightarrow$  **strong scalability**)
- ❖ Илүү олон процессор ашиглах үед бид илүү том хэмжээтэй асуудлыг авч үзнэ. ( $\Rightarrow$  **weak scalability**)
  - ❖ Энэ тохиолдолд параллел хэсэгт зарцуулах хугацаа ( $T_{\text{par}}$ ) нь  $T_{\text{ser}}$  тэй харьцуулахад хуран өсч болно
- ❖ **Scaled Speedup:** хүрч болох speedup-ыг тооцоолоходоо дээрх нөхцлийг багтаасан болно.
- ❖ Одоо бид асуудлын хүндрэлийг харгалзсан илүү ерөнхий хуулийг олно.
  - ❖ **Gustafson-ийн хууль** нь дараалсан хэсэг нь тогтмол бөгөөд параллел хэсэг нь асуудлын хэмжээнээс шугаман хамаарч өсдөг үед олон процессор ашиглахад онолын хувьд хүрч болох speedup-г урьдчилан таамаглахад хэрэглэж болох тусгай тохиолдол юм.

# Scaled Speedup хуулийн гаргалгаа

$$T_{\alpha\beta}(1) = \alpha \cdot T_{\text{ser}} + \beta \cdot T_{\text{par}} = \alpha \cdot f \cdot T(1) + \beta \cdot (1 - f) \cdot T(1)$$

$\alpha$ : асуудлын хэмжээний хүндрэлтэй параллелчлалд хамаарахгүй програмын хэсгийн өсөлтийн функц

$\beta$ : асуудлын хэмжээний хүндрэлтэй параллелчлалд хамаарах програмын хэсгийн өсөлтийн функц

- ❖ Боломжит speedup-н scaled дээд хязгаарыг тодорхойлно:

$$\begin{aligned} S_{\alpha\beta}(p) &= \frac{T_{\alpha\beta}(1)}{T_{\alpha\beta}(p)} \leq \frac{\alpha \cdot f \cdot T(1) + \beta \cdot (1 - f) \cdot T(1)}{\alpha \cdot f \cdot T(1) + \frac{\beta \cdot (1 - f) \cdot T(1)}{p}} \\ &= \frac{\alpha \cdot f + \beta \cdot (1 - f)}{\alpha \cdot f + \frac{\beta \cdot (1 - f)}{p}} \end{aligned}$$

# Gustafson-ы хуулийн гаргалгаа

$$\gamma = \frac{\beta}{\alpha}$$

Параллеллагдах болон параллеллагдахгүй хэсгийн хорондох асуудлын хүндрэлийн харьцаа.

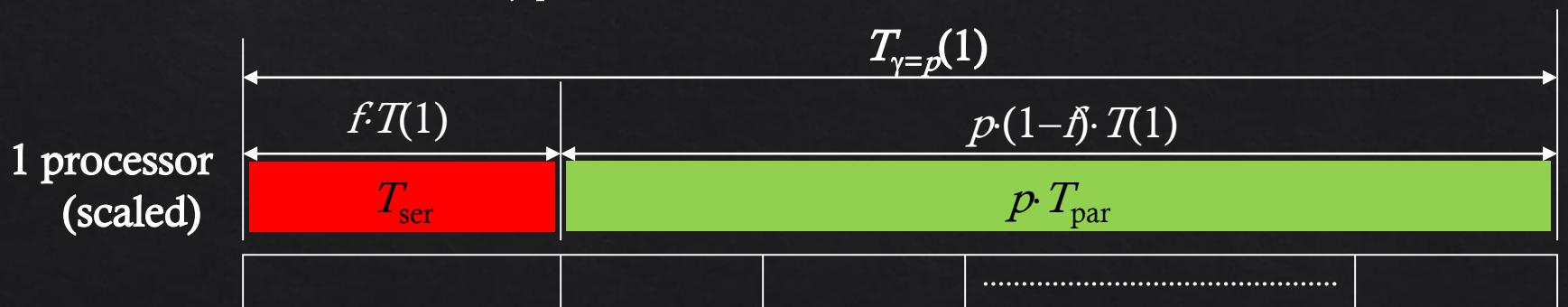
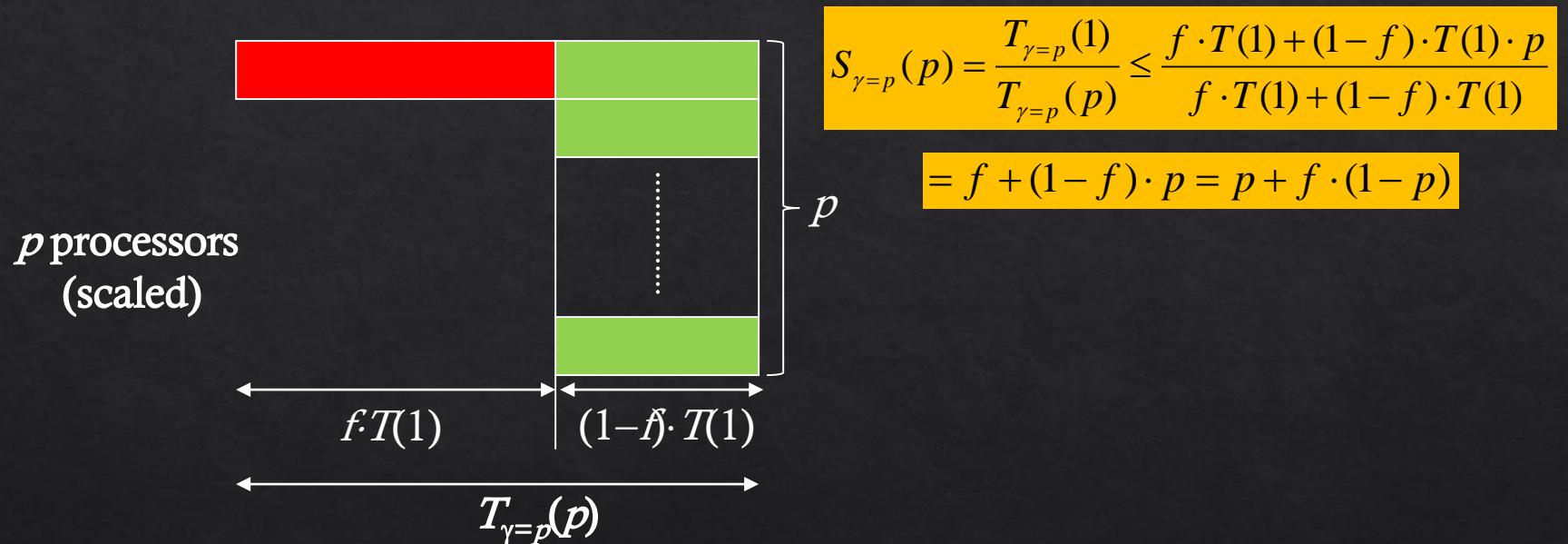
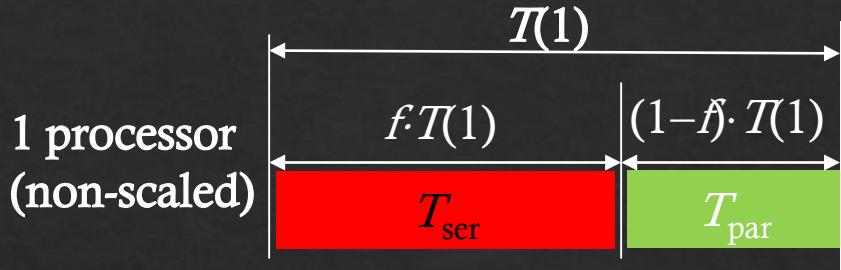
$$S_\gamma(p) \leq \frac{f + \gamma \cdot (1 - f)}{f + \frac{\gamma \cdot (1 - f)}{p}}$$

Дараах тусгай тохиолдлуудад  $\gamma$ -ийн хэлбэрүүдийг хэрэглэглэнэ:

1.  $\gamma = 1$  (жны:  $\alpha = \beta$ ): **Amdahl-ын хууль**
2.  $\gamma = p$  (жны:  $\alpha = 1$ ;  $\beta = p$ ): Параллеллагдахгүй хэсэг тогтмол байхад параллеллагдах хэсэг  $p$ -ээс хамаарч шугаман өсдөг.  $\Rightarrow$  **Gustafson-ы хууль**:

$$S(p) \leq f + p \cdot (1 - f) = p + f \cdot (1 - p)$$

# Gustafson-ы хууль



# Scaled Speedup – Жишээ

- ❖ 15% нь дараалсан, 85% нь асуудлын өгөгдсөн хэмжээнээс шугаман хамааралтай параллеллагдах хэсгүүдтэйй програмыг параллелаар ажиллуулна. Асуудлын хэмжээ хязгаарлагдах тул дараалсан хэсгийн цаг өсөхгүй гэж бодъё.
  - i. Хэрэв асуудал өсөхгүйгээр 50 процессор ашигладаг бол хичнээн speedup болох вэ?

$$S_{\gamma=1}(50) \leq \frac{f + \gamma \cdot (1 - f)}{f + \frac{\gamma \cdot (1 - f)}{p}} = \frac{1}{0.15 + \frac{0.85}{50}} = 5.99$$

- ii. Бид асуудлын хэмжээ 100 дахин өссөн гэж үзье. 50 процессортой байхад хичнээн speedup болох вэ?

$$S_{\gamma=100}(50) \leq \frac{f + \gamma \cdot (1 - f)}{f + \frac{\gamma \cdot (1 - f)}{p}} = \frac{0.15 + 100 \cdot 0.85}{0.15 + \frac{100 \cdot 0.85}{50}} = 46.03$$

# Дасгал

- ❖ 128 процессор дээр 100-ийн speedup хүрэх ёстой програм бичихийг хүсч байна гэж бодъё.
  - i. Strong scalability-н таамаглал дээр speedup очих үед программын хамгийн их дараалсан хэсэг ямар байх вэ?

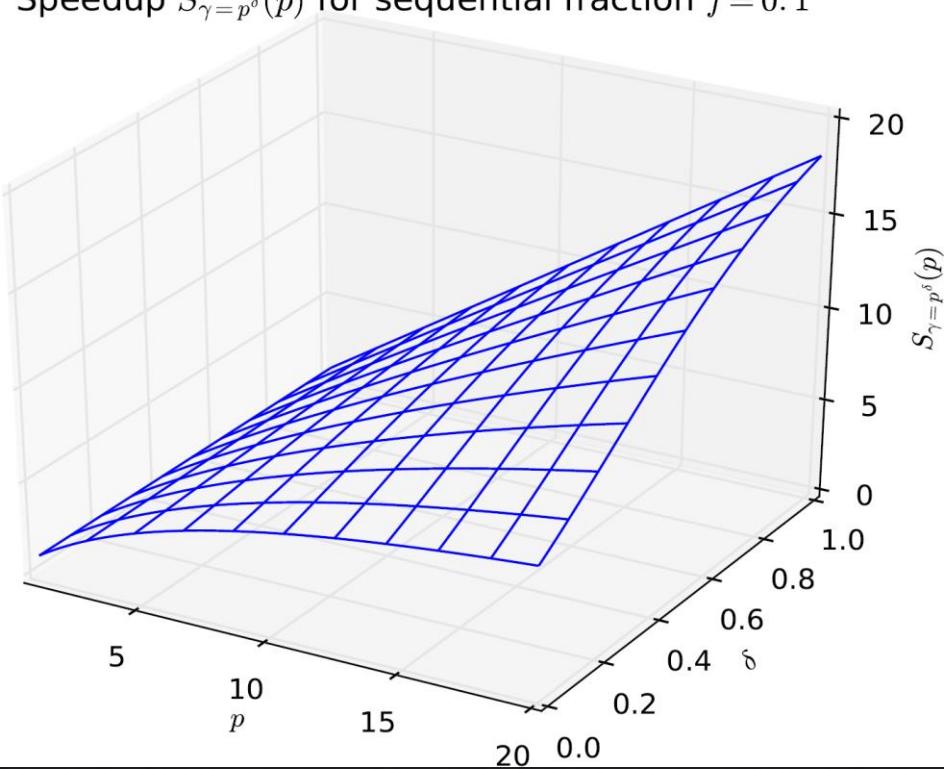
$$100 = \frac{1}{f + \frac{(1-f)}{128}} = \frac{128}{128 \cdot f + 1 - f} = \frac{128}{127 \cdot f + 1} \Rightarrow f = 0.0022$$

- ii. γ харьцаа шугаман өсч байхад weak scalability-н таамаглал дээр speedup очих үед программын хамгийн их дараалсан хэсэг ямар байх вэ?

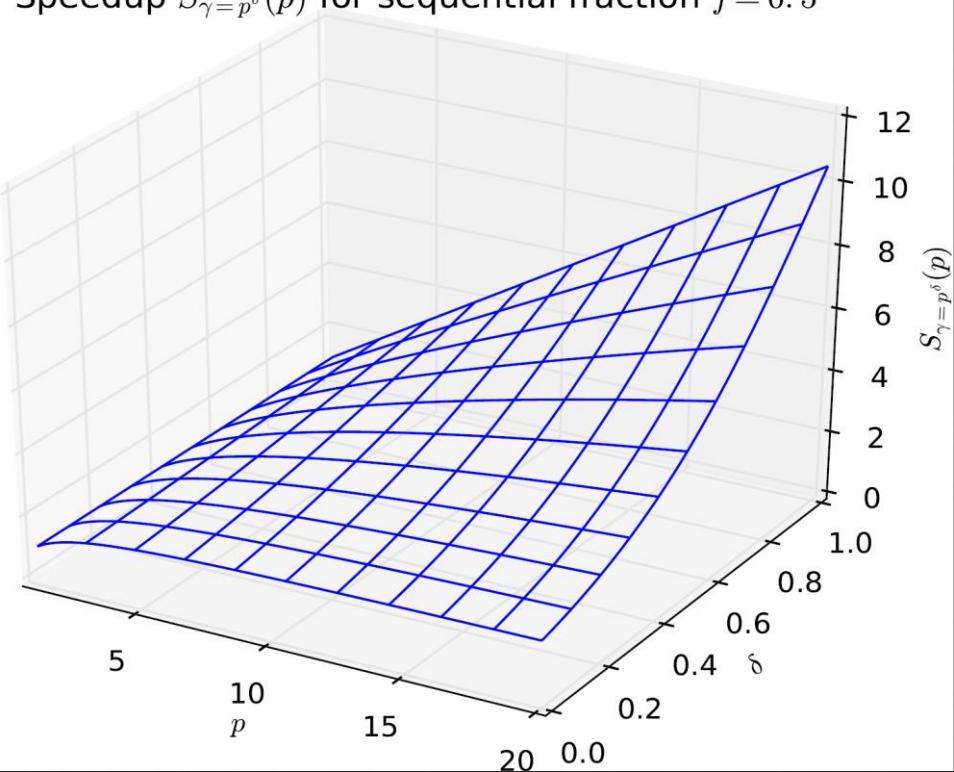
$$100 = 128 + f \cdot (1 - 128) = 128 - 127 \cdot f \Rightarrow f = \frac{28}{127} = 0.22$$

# $\gamma=p^\delta$ параметртэй Scaled Speedup $S_\gamma(p)$ -ийн функционал хамаарал

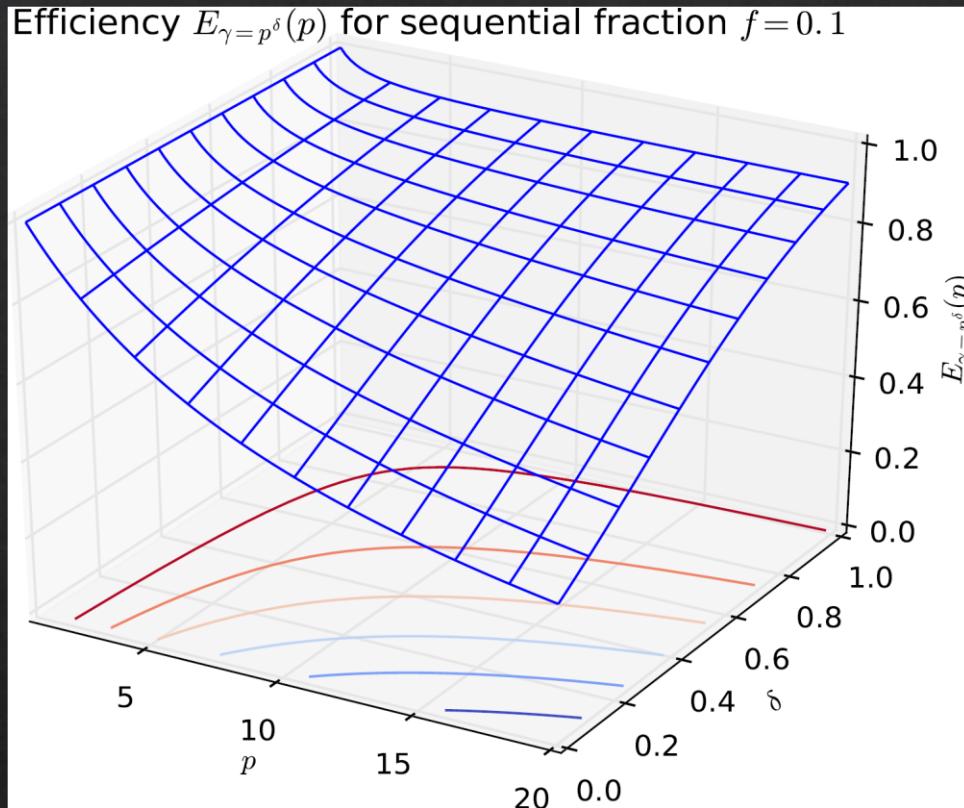
Speedup  $S_{\gamma=p^\delta}(p)$  for sequential fraction  $f=0.1$



Speedup  $S_{\gamma=p^\delta}(p)$  for sequential fraction  $f=0.5$



$\gamma = p^\delta$  параметртэй Scaled Efficiency  
 $E_\gamma(p) = S_\gamma(p)/p$ -н функционал хамаарал

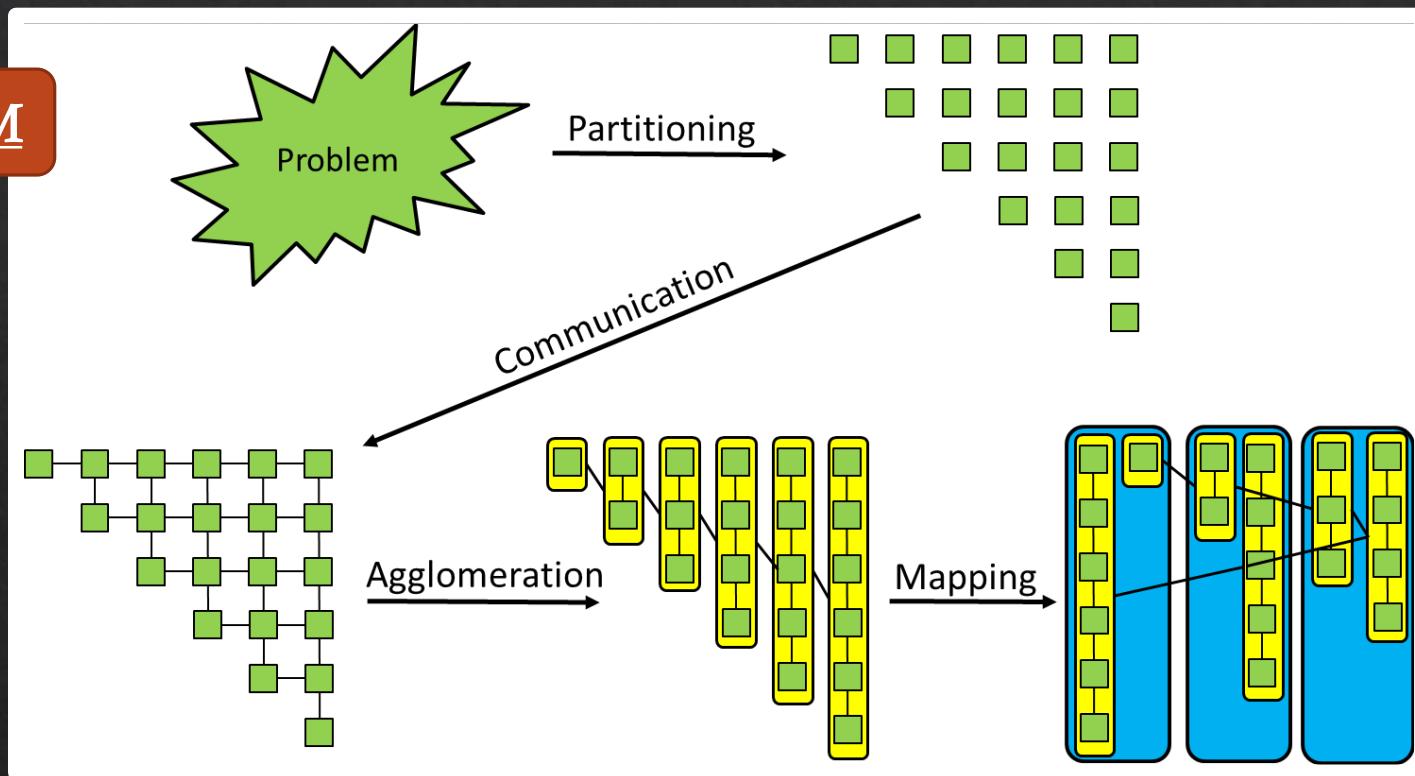


Iso-efficiency шугам:  $p$  олон болоход параллел үр ашгийг хадгалахын тулд  $\delta$  өсөлт хир байх вэ? эсвэл:

Хэрэв бид илүү олон тооцолох нэгжүүд хэрэглэхэд ижил үр ашгийг хүсч байвал манай асуудлын параллеллагдах хэсэг хир зэрэг байх вэ?

# Foster's Parallel Algorithm Design Method

PCAM



1. **Partitioning:** асуудлыг параллел ажилуулж болох олон жижиг даалгавруудад (*fine-grained*) задлана.
2. **Communication:** даалгавруудын хоорондох шаардлагатай мэдээлэл солилцоог тодорхойлох
3. **Agglomeration:** Өгөгдлийн байршлыг сайжруулан мэдээлэл солилцоог багасгах зорилгоор тодорхойлсон даалгаваруудыг том даалгаварт нэгтгэнэ
4. **Mapping:** Мэдээлэл солилцоог багасгах, давхардалгүй байдлыг хангах болон ачааллыг тэнцвэржүүлэх зорилгоор бөөгнөрлийг процесс/thread-үүдад хуваарилна

# Жишээ: Jacobi Iteration

- ❖ Матриц дахь утга бүрийг дөрвөн хөршийнх дундаж утгаар солино
  - ❖ Хүрээний утгууд нь тогтмол байна

Оролтын матриц:  $x[i][j]$

$xnew[i][j]$

# Jacobi Iteration

- ❖ Конвергенци болох хүртэлх давталтын алхам бүрт өгөгдсөн 2D матрицын бүх цэгийн утгыг хөршүүдийнх нь дундажаар солино.

```
while (not converged) {  
    for (int i=1; i<rows-1; i++)  
        for (int j=1; j<cols-1; j++)  
            buff[i*cols+j] = 0.25f*(data[(i+1)*cols+j] + data[i*cols+j-1]  
                + data[i*cols+j+1] + data[(i-1)*cols+j]);  
  
    memcpy(data,buff,rows*cols*sizeof(float));  
}
```

- Хүрээний утгуудыг ялагсан:

```
x[0][j]  
x[n-1][j]  
x[i][0]  
x[i][n-1]
```

# Жишээ: Jacobi Iteration

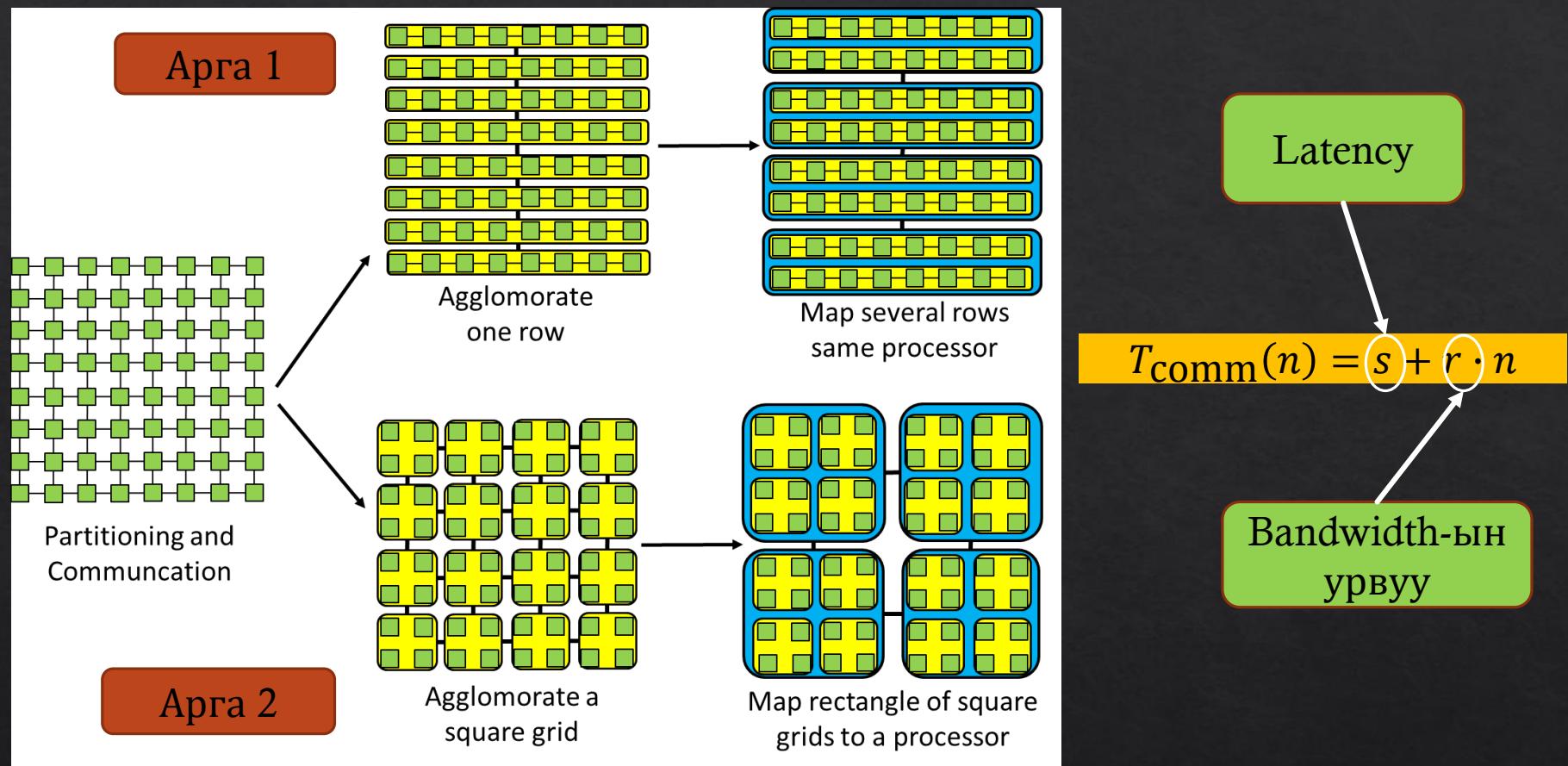
25 давталын дараах  $x[i][j]$

**75** давталын дараах  $x[i][j]$

# Agglomeration схемүүд

- ❖ Method 1: Бүх даалгаврууд ижил мөрөнд байдаг. Хэд хэдэн мөрийг нэгтгэн нэг том даалгавар болгоно.
- ❖ Method 2: Хэд хэдэн даалгавар дөрвөлжин нэгжид байдаг. Хэд хэдэн хөрш дөрвөлжингүүдийг нэгтгэн нэг том даалгавар болгоно.

# Jacobi Iteration-ын хоёр схем



- ❖ **Арга 1:** Хоёр процессорын хоорондох мэдээлэл солилцоо  $\approx 2(s + r \cdot n)$
- ❖ **Арга 2:** Хоёр процессорын хоорондох мэдээлэл солилцоо  $\approx 4 \left( s + r \left( \frac{n}{\sqrt{p}} \right) \right)$
- ❖  $\Rightarrow$  Арга 1-д тогтмол байгаа мэдээлэл солилцооны хугацаа нь  $p$ -ийн их утганд багасч байгаагаара Арга 2 давуу талтай байна.

Баярлалаа.