Práctica - 1 *Slurm, BWA y Samtools*

David Barcene david.barcene@utp.ac.pa

24 de febrero de 2026

Objetivo: Ejecutar el programa *BWA* con un set de datos reales reproduciendo los resultados, para poder entender como se ejecutan los comandos con *Slurm*.

1. Directorios

El directorio rincipal ˜/**data** consta de 3 subdirectorios:

- **reads**: Datos de seccuenciación, 12 muestras en sus correspondientes archivos.
- **ref**: Genomas de referencia indexados.
- **BAM**: directorio output

2. Procedimiento

1. Ejecutar el programa **BWA** para alinear la muestra al genoma de referencia.f En cada paso reemplazar la palabra "muestraçon el nombre de la muestra (ej. **BAL_C1_3h**, **BAL_C2_3h**, etc.).

```
bwa mem -t 30 -o BAM/BAL_C1_3h_LpmP.sam \
ref/LpmP_2025_union.fasta \
reads/muestra_1.fq \
reads/muestra_2.fq
```

2. Ejecutar el programa **samtools** en tres pasos para generar un fichero binario de alineamiento en formato **BAM**

```
# Paso 1: view
samtools view -b -@ 30 \
-o BAM/BAL_C1_3h_LpmP_unsorted.bam\
BAM/BAL_C1_3h_LpmP.sam

# Paso 2: sort
samtools sort -b -@ 30 \
-o BAM/BAL_C1_3h_LpmP.bam\
BAM/BAL_C1_3h_LpmP_unsorted.sam

# Paso 3: index
samtools index BAM/BAL_C1_3h_LpmP.bam
```

3. Ejecutar el programa **samtools** nuevamente para guardar las estadísticas de alineamiento

```
samtools flagstat -@ 30 BAM/BAL_C1_3h_LpmP,bam >
muestra_stats.txt
```

Nota: Las opciones **bwa -t** y **samtools -**@ indican el número de núcleos. Este debe coincidir con el que se le solicite a Slurm via #**SBATCH** –**cpus-per-task**. El máximo numero de núcleos por nodo es 40.

Listing 1: Directory Tree

```
data
|____BAM
|____ref
| |____LpmP_2025_union.fasta
| |____LpmP_2025_union.fasta.amb
| |____LpmP_2025_union.fasta.ann
| |____LpmP_2025_union.fasta.bwt
| |____LpmP_2025_union.fasta.pac
| |____LpmP_2025_union.fasta.sa
|____reads
| |____BAL_C1_3h_1.fq
| |____BAL_C1_3h_2.fq
| |____BAL_C2_3h_1.fq
| |____BAL_C2_3h_2.fq
| |____BAL_C3_3h_1.fq
| |____BAL_C3_3h_2.fq
| |____BAL_I1_3h_1.fq
| |____BAL_I1_3h_2.fq
| |____BAL_I2_3h_1.fq
| |____BAL_I2_3h_2.fq
| |____BAL_I3_3h_1.fq
| |____BAL_I3_3h_2.fq
| |____BL6_C1_3h_1.fq
| |____BL6_C1_3h_2.fq
| |____BL6_C2_3h_1.fq
| |____BL6_C2_3h_2.fq
| |____BL6_C3_3h_1.fq
| |____BL6_C3_3h_2.fq
| |____BL6_I1_3h_1.fq
| |____BL6_I1_3h_2.fq
| |____BL6_I2_3h_1.fq
| |____BL6_I2_3h_2.fq
| |____BL6_I3_3h_1.fq
| |____BL6_I3_3h_2.fq
```

3. Resultados

Inicialmente se redactaron tres scripts para ejecutar 3 pasos de procesamiento sobre todas las muestras de forma secuencial sobre un solo nodo. Esta solución implica que se debe esperar a que se ejecuten todos los pasos sobre una sola muestra para poder continuar con la siguiente.

3.1. Paralelización

Se redactaron dos scripts, el primero llamado **sample_proc.sh** que contiene todos los pasos para el tratamiento de una sola muestra en un solo nodo. En este script se colocan los comentarios #**SBATCH** luego del shebang y los programas **bwa** y **samtools** se corren utilizando el comando **srun** de **SLURM** para que sean gestionados.

```
#!/bin/bash
#SBATCH --job-name=bio_pipe
#SBATCH --cpus-per-task=30

# The sample name is passed as the first argument to the script
if [ -z "$1"  ]; then
            echo "Error [1]: No sample name provided."
            exit 1
fi

# Path variables
READS_DIR="./reads"
OUTPUT_DIR="./BAM"
REF="./ref/LpmP_2025_union.fasta"

# Make output dir if doesn't exist yet
mkdir -p $OUTPUT_DIR

MUESTRA=$1
echo "--- Starting Processing for: ${MUESTRA} ---"

################################## BWA ########################################
# Step 0: BWA
echo "Step 0: Mapping with BWA"

srun bwa mem -t $SLURM_CPUS_PER_TASK \
        -o ${OUTPUT_DIR}/${MUESTRA}_LpmP.sam \
        ${REF} \
        ${READS_DIR}/${MUESTRA}_1.fq \
        ${READS_DIR}/${MUESTRA}_2.fq
```

Nótese que luego de **-t** no se define un número directo de cpus, en vez se llama la variable interna **$SLURM_CPUS_PER_TASK**, la cual está definida por el comando **#SBATCH –cpus-per-task=30**

El uso de este script tiene la siguiente sintaxis:

```
./sample_proc.sh sample_name
```

```
################################ SAMTOOLS #####################################
# Step 1: Samtools View
echo "Step 1: Samtools View"

srun samtools view -b -@ $SLURM_CPUS_PER_TASK \
        -o ${OUTPUT_DIR}/${MUESTRA}_LpmP_unsorted.bam \
        ${OUTPUT_DIR}/${MUESTRA}_LpmP.sam



# Step 2: Samtools Sort
echo "Step 2: Sorting BAM"

srun samtools sort -@ $SLURM_CPUS_PER_TASK \
```

```
        -o ${OUTPUT_DIR}/${MUESTRA}_LpmP.bam \
        ${OUTPUT_DIR}/${MUESTRA}_LpmP_unsorted.bam



# Step 3: Samtools Index
echo "Step 3: Indexing"
srun samtools index -@ $SLURM_CPUS_PER_TASK \
        ${OUTPUT_DIR}/${MUESTRA}_LpmP.bam



# Step 4: Samtools Stats
echo "Step 4: Generating Stats"
srun --output=${OUTPUT_DIR}/${MUESTRA}_stats.txt samtools flagstat -@
    $SLURM_CPUS_PER_TASK \
        ${OUTPUT_DIR}/${MUESTRA}_LpmP.bam

echo "--- Finished Processing for: $MUESTRA ---"
```

Nota: Para los outputs de cada programa de la suite **samtools** se debe revisar si tienen salida en stdout (standard output). **samtools flagstat** es un ejemplo claro de un programa sin salida en stdout, por lo cual se le indica a **srun** el archivo de salida mediante la opción **–outpput**.

Y el segundo script **run_sbatch.sh** es un orquestrador que actúa como lanzador principal, automatizando el envío de múltiples tareas al cluster mediante el comando **sbatch** de **SLURM** para enviar el primer **run_sbatch.sh** a la cola del cluster. **SLURM** se encargará de distribuir las tareas en los nodos disponibles. En el caso de que hayan más tareas que nodos disponibles, las tareas extra quedaran pendientes en cola de espera.

```
#!/bin/bash
READS_DIR="reads"

for READ1 in ${READS_DIR}/*_1.fq; do
        # Strip filename to get sample name
        FILENAME=$(basename "$READ1")
        MUESTRA=${FILENAME%_1.fq}

        echo "Submitting␣job␣for␣sample:␣${MUESTRA}"

        # Launch the processing script and pass the sample
        # name as an argument
        sbatch --job-name="$MUESTRA" sample_proc.sh "$MUESTRA"
done
```

Pra poder revisar que tarea se está ejecutando en el cluster se pueden utilizar las opciones **sacct** que muestra información de la base de datos de **SLURM**, o mediante **squeue -u username** para ver que tareas ejecuta nuestro usuario. Adicionalmente se colocó la opción **srun –job-name=$MUESTRA**, mediante la cual se le coloca le nombre de muestra a cada tarea ejecutada por el orquestrador.